```
/* +++Date last modified: 05-Jul-1997 */

/*
** A quick "life" (2-d cellular automaton) implementation done in Turbo C 2.0
** on the spur-of-the-moment by Jonathan Guthrie 9/20/1992 and donated to the
** public domain.
**
** In keeping with the guidelines of the C_ECHO, this program has been tested,
** and does seem to operate properly.
*/

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

#ifndef random
 #define random(num) (int)(((long)rand()*(num))/RAND_MAX)
#endif

/*
**  From VIDPORT.C, also in SNIPPETS
*/

void GotoXY(int col, int row);
void ClrScrn(int vattrib);

#ifndef randomize
 #define randomize() srand(((unsigned int)time(NULL))|1)
#endif

#define     ROWS        24
#define     COLS        80
#define     GENERATIONS 10

int civ1[ROWS][COLS], civ2[ROWS][COLS];

void update_generation(int old[ROWS][COLS], int new[ROWS][COLS])
{
      int i, j, count;

      for (i = 0; i < ROWS; ++i)
      {
            for (j = 0; j < COLS; ++j)
            {
                  count = old[(i + ROWS - 1) % ROWS][(j + COLS - 1) % COLS] +
                        old[(i + ROWS - 1) % ROWS][j] +
                        old[(i + ROWS - 1) % ROWS][(j + 1) % COLS] +
                        old[i][(j + COLS - 1) % COLS] +
                        old[i][(j + 1) % COLS] +
                        old[(i + 1) % ROWS][(j + COLS - 1) % COLS] +
                        old[(i + 1) % ROWS][j] +
                        old[(i + 1) % ROWS][(j + 1) % COLS];

                  switch(count)
                  {
                  case 0:
                  case 1:
                  case 4:
                  case 5:
                  case 6:
                  case 7:
                  case 8:
                        new[i][j] = 0;
                        break;

                  case 2:
                        new[i][j] = old[i][j];
                        break;

                  case 3:
                        new[i][j] = 1;
                        break;
                  }

                  GotoXY(j+1, i+1);
                  putch(new[i][j] ? '*' : ' ');
            }
      }
}


void initialize(void)
{
      int i, j;

      randomize();
      ClrScrn(7);

      for (i = 0; i < ROWS; ++i)
      {
            for (j = 0; j < COLS; ++j)
            {
                  civ1[i][j] = random(2);
                  GotoXY(j+1, i+1);
                  putch(civ1[i][j] ? '*' : ' ');
            }
      }
}

```

```
 101 |
 102 | int main(void)
 103 | {
 104 |       int i;
 105 |
 106 |       initialize();
 107 |       for (i = 0; i < GENERATIONS; ++i)
 108 |       {
 109 |             update_generation(civ1, civ2);
 110 |             update_generation(civ2, civ1);
 111 |       }
 112 |       return EXIT_SUCCESS;
 113 | }
```

TEXT STATISTICS

```
  2791 characters
   113 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     4 constants [character]
     0 constants [string]
    38 constants [numeric]
```

SYMBOL TABLE

```
    COLS            33      36+     38+     44      46+     48      49+     50      51+     53      92
    ClrScrn         26      88
    EXIT_SUCCESS           112
    GENERATIONS             34     107
    GotoXY          25      76      95
    NULL            29
    RAND_MAX        18
    ROWS            32      36+     38+     42      46+     47+     48+     51      52      53      90
    civ1            36      94      96     109     110
    civ2            36     109     110
    col             25
    conio           13
    count           40      46      55
    h       12      13      14      15
    i       40      42+     46      47      48      49      50      51      52      53      64      68+
            72      76      77      85      90+     94      95      96     104     107+
    initialize              83     106
    j       40      44+     46      47      48      49      50      51      52      53      64      68+
            72      76      77      85      92+     94      95      96
    main           102
    num             18+
    old             38      46      47      48      49      50      51      52      53      68
    putch           77      96
    rand            18
    random          17      18      94
    randomize       28      29      87
    row             25
    srand           29
    stdio           12
    stdlib          14
    time            15      29
    update_generation       38     109     110
    vattrib         26
```

```
/* +++Date last modified: 05-Jul-1997 */

/*

AMALLOC - multi-dimensional malloc()

Allocates a multidimensional array dynamically, at runtime, so that
  1: its elements can be accessed using multiple indirection
  2: it can be deallocated using a call to the standard free() function
Note: On PC's the max array size is 64K

Paul Schlyter, 1992-02-09.  Released to the public domain.

*/

#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "snparray.h"


#define MAXDIMS  5              /* Defines the maximum number of dimensions */
#define MAXSIZE  ((size_t) -1L) /* Maximum size of array */


void *amalloc( int esiz, void *initval, int dims, ... )
/*
 *  Input:   esiz     size of each array elements, as given by sizeof
 *           initval  pointer to initial value. NULL ==> zero fill
 *           dims     number of dimensions: 1..MAXDIMS (5)
 *           ...      number of elements in each dimension (int's)
 *
 *  Returns:  NULL    error: out of memory, or illegal parameters
 *                    otherwise base pointer to array
 */
{
      unsigned int dim[MAXDIMS], accdim[MAXDIMS];
      va_list ap;
      int i, j;
      long int totsiz;
      void **q;
      char *p, *r, *s;

      if (dims < 1  ||  dims > MAXDIMS)
            return NULL;

      memset(dim, 0, sizeof(dim));          /* Read dimension numbers */
      memset(accdim, 0, sizeof(accdim));
      va_start(ap, dims);
      dim[0] = accdim[0] = va_arg(ap,int);
      for (i = 1; i < dims; i++)
      {
            dim[i] = va_arg(ap,int);
            accdim[i] = accdim[i-1] * dim[i];
      }
      va_end(ap);

                                            /* Compute total array size */
      totsiz = esiz * accdim[dims-1];       /* Data size */

      for (i = 0; i < dims - 1; i++ )       /* Add space for pointers */
            totsiz += sizeof(void *) * accdim[i];

      if (totsiz > MAXSIZE)                 /* Exit if totsiz too large */
            return NULL;

      p = malloc((size_t) totsiz);          /* Allocate memory */
      if (p == NULL)                        /* Out-of-memory   */
            return NULL;
      memset(p, 0, (unsigned int) totsiz);  /* Zero out allocated memory */
      q = (void **) p;

      if (dims == 1)
            r = (char *) q + esiz * accdim[0];

      for (i = 1; i < dims; i++)            /* Fill in pointers */
      {
            int siz;
            int accd = accdim[i-1], d = dim[i];

            siz =  i == dims-1 ? esiz : sizeof(void *);

            r = (char *) q + sizeof(void *) * accd;
            for (j = 0; j < accd; j++)
            {
                  *q++ = r;
                  r += siz * d;
            }
      }

      if (initval != NULL)
      {
            for (s = (char *) q; s < r; s += esiz)
                  memcpy(s, initval, esiz);
      }

      return p;

}  /* amalloc */
```

```
 101 |
 102 | #ifdef TEST   /* Test program */
 103 |
 104 | #include <stdio.h>
 105 |
 106 | main()
 107 | {
 108 |       static char init_d[8] = { 0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF };
 109 |       int init_i = 0x1111;
 110 |       double *a   = amalloc( sizeof(double), init_d, 1, 4 );
 111 |       double **b  = amalloc( sizeof(double), init_d, 2, 4, 5 );
 112 |       double ***c = amalloc( sizeof(double), init_d, 3, 4, 5, 6 );
 113 |       int ***d = amalloc( sizeof(int), &init_i, 3, 4, 5, 6 );
 114 |       int i, j, k;
 115 |
 116 |       for (i = 0; i < 4; i++)
 117 |             for (j = 0; j < 5; j++ )
 118 |                   for (k = 0; k < 6; k++ )
 119 |                         d[i][j][k] = (i * 256) + (j * 16) + k;
 120 |
 121 |       a = a, b = b, c = c;
 122 |
 123 |       return 0;
 124 | }
 125 |
 126 | #endif /* TEST */
```

TEXT STATISTICS

```
  3669 characters
   126 lines
```

LEXICAL STATISTICS

```
    17 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     1 constants [string]
    51 constants [numeric]
```

SYMBOL TABLE

```
    MAXDIMS         22      37+     44
    MAXSIZE         23      64
    NULL            45      65      68      69      91
    TEST           102
    a      110     121+
    accd            79      83      84
    accdim          37      48+     50      54+     59      62      74      79
    amalloc         26     110     111     112     113
    ap              38      49      50      53      56
    b      111     121+
    c      112     121+
    d       79      87     113     119
    dim             37      47+     50      53      54      79
    dims            26      44+     49      51      59      61      73      76      81
    esiz            26      59      74      81      93      94
    h       16      17      18     104
    i       39      51+     53      54+     61+     62      76+     79+     81     114     116+    119+
    init_d         108     110     111     112
    init_i         109     113
    initval         26      91      94
    j       39      84+    114     117+    119+
    k      114     118+    119+
    main           106
    malloc          67
    memcpy          94
    memset          47      48      70
    p       42      67      68      70      71      97
    q       41      71      74      83      86      93
    r       42      74      83      86      87      93
    s       42      93+     94
    siz             78      81      87
    size_t          23      67
    stdarg          17
    stdio          104
    stdlib          16
    string          18
    totsiz          40      59      62      64      67      70
    va_arg          50      53
    va_end          56
    va_list         38
    va_start        49
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FLENGTH.C - a simple function using all ANSI-standard functions
**              to determine the size of a file.
**
**  Public domain by Bob Jarvis.
*/

#include "snipfile.h"

#if defined(__cplusplus) && __cplusplus /* C++ version follows */

 #include <fstream.h>

 long flength(char *fname)
 {
       long length = -1L;
       ifstream ifs;

       ifs.open(fname, ios::binary);
       if (ifs)
       {
             ifs.seekg(0L, ios::end) ;
             length = ifs.tellg() ;
       }
       return length;
 }

#else /* Straight C version follows */

 long flength(char *fname)
 {
       long length = -1L;
       FILE *fptr;

       fptr = fopen(fname, "rb");
       if(fptr != NULL)
       {
             fseek(fptr, 0L, SEEK_END);
             length = ftell(fptr);
             fclose(fptr);
       }

       return length;
 }

#endif /* C++ */

#ifdef TEST

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif
#ifdef __TURBOC__
 #pragma argsused
#endif

main(int argc, char *argv[])
{
      char *ptr;
      long len;

      while (--argc)
      {
            len = flength(ptr = *(++argv));
            if (-1L == len)
                  printf("\nUnable to get length of %s\n", ptr);
            else  printf("\nLength of %s = %ld\n", ptr, len);
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1428 characters
    74 lines

LEXICAL STATISTICS

     6 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     4 constants [string]
     6 constants [numeric]

SYMBOL TABLE

    FILE            35
    NULL            38
    SEEK_END        40
    TEST            50
    __TURBOC__              55
    __WATCOMC__             52
    __cplusplus             12+
    argc            59      64
    argsused        56
    argv            59      66
    binary          21
    defined         12
    end             24
    fclose          42
    flength         16      32      66
    fname           16      21      32      37
    fopen           37
    fptr            35      37      38      40      41      42
    fseek           40
    fstream         14
    ftell           41
    h       14
    ifs             19      21      22      24      25
    ifstream        19
    ios             21      24
    len             62      66      67      69
    length          18      25      27      34      41      45
    main            59
    off             53
    open            21
    printf          68      69
    ptr             61      66      68      69
    seekg           24
    tellg           25
    unreferenced            53
```

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************
 *
 * Fuzzy string searching subroutines
 *
 * Author:    John Rex
 * Date:      August, 1988
 * References: (1) Computer Algorithms, by Sara Baase
 *                 Addison-Wesley, 1988, pp 242-4.
 *             (2) Hall PAV, Dowling GR: "Approximate string matching",
 *                 ACM Computing Surveys, 12:381-402, 1980.
 *
 * Verified on:
 *    Datalite, DeSmet, Ecosoft, Lattice, MetaWare, MSC, Turbo, Watcom
 *
 * Compile time preprocessor switches:
 *    TEST - if defined, include test driver
 *
 * Usage:
 *
 *    char *pattern, *text;  - search for pattern in text
 *    int degree;      - degree of allowed mismatch
 *    char *start, *end;
 *    int howclose;
 *
 *    void App_init(pattern, text, degree);   - setup routine
 *    void App_next(&start, &end, &howclose); - find next match
 *
 *    - searching is done when App_next() returns start==NULL
 *
 *************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "phonetic.h"

/* local, static data */

static char *Text, *Pattern; /* pointers to search strings       */
static int Textloc;          /* current search position in Text  */
static int Plen;             /* length of Pattern                */
static int Degree;           /* max degree of allowed mismatch   */
static int *Ldiff, *Rdiff;   /* dynamic difference arrays        */
static int *Loff,  *Roff;    /* used to calculate start of match */

void App_init(char *pattern, char *text, int degree)
{
      int i;

      /* save parameters */

      Text = text;
      Pattern = pattern;
      Degree = degree;

      /* initialize */

      Plen = strlen(pattern);
      Ldiff  = (int *) malloc(sizeof(int) * (Plen + 1) * 4);
      Rdiff  = Ldiff + Plen + 1;
      Loff   = Rdiff + Plen + 1;
      Roff   = Loff +  Plen + 1;
      for (i = 0; i <= Plen; i++)
      {
            Rdiff[i] = i;   /* initial values for right-hand column */
            Roff[i]  = 1;
      }

      Textloc = -1; /* current offset into Text */
}

void App_next(char **start, char **end, int *howclose)
{
      int *temp, a, b, c, i;

      *start = NULL;
      while (*start == NULL)  /* start computing columns */
      {
            if (Text[++Textloc] == '\0') /* out of text to search! */
                  break;

            temp = Rdiff;   /* move right-hand column to left ... */
            Rdiff = Ldiff;  /* ... so that we can compute new ... */
            Ldiff = temp;   /* ... right-hand column */
            Rdiff[0] = 0;   /* top (boundary) row */

            temp = Roff;    /* and swap offset arrays, too */
            Roff = Loff;
            Loff = temp;
            Roff[1] = 0;

            for (i = 0; i < Plen; i++)    /* run through pattern */
            {
                  /* compute a, b, & c as the three adjacent cells ... */

                  if (Pattern[i] == Text[Textloc])
                        a = Ldiff[i];
                  else  a = Ldiff[i] + 1;
```

```
101 |                   b = Ldiff[i+1] + 1;
102 |                   c = Rdiff[i] + 1;
103 |
104 |                   /* ... now pick minimum ... */
105 |
106 |                   if (b < a)
107 |                         a = b;
108 |                   if (c < a)
109 |                         a = c;
110 |
111 |                   /* ... and store */
112 |
113 |                   Rdiff[i+1] = a;
114 |             }
115 |
116 |             /* now update offset array */
117 |             /* the values in the offset arrays are added to the
118 |                current location to determine the beginning of the
119 |                mismatched substring. (see text for details) */
120 |
121 |             if (Plen > 1) for (i=2; i<=Plen; i++)
122 |             {
123 |                   if (Ldiff[i-1] < Rdiff[i])
124 |                         Roff[i] = Loff[i-1] - 1;
125 |                   else if (Rdiff[i-1] < Rdiff[i])
126 |                         Roff[i] = Roff[i-1];
127 |                   else if (Ldiff[i] < Rdiff[i])
128 |                         Roff[i] = Loff[i] - 1;
129 |                   else /* Ldiff[i-1] == Rdiff[i] */
130 |                         Roff[i] = Loff[i-1] - 1;
131 |             }
132 |
133 |             /* now, do we have an approximate match? */
134 |
135 |             if (Rdiff[Plen] <= Degree)    /* indeed so! */
136 |             {
137 |                   *end = Text + Textloc;
138 |                   *start = *end + Roff[Plen];
139 |                   *howclose = Rdiff[Plen];
140 |             }
141 |       }
142 |
143 |       if (start == NULL) /* all done - free dynamic arrays */
144 |             free(Ldiff);
145 | }
146 |
147 | #ifdef TEST
148 |
149 | main(int argc, char **argv)
150 | {
151 |       char *begin, *end;
152 |       int howclose;
153 |
154 |       if (argc != 4)
155 |       {
156 |             puts("Usage is: approx pattern text degree\n");
157 |             exit(0);
158 |       }
159 |
160 |       App_init(argv[1], argv[2], atoi(argv[3]));
161 |       App_next(&begin, &end, &howclose);
162 |       while (begin != NULL)
163 |       {
164 |             printf("Degree %d: %.*s\n", howclose, end-begin+1, begin);
165 |             App_next(&begin, &end, &howclose);
166 |       }
167 |       return 0;
168 | }
169 |
170 | #endif /* TEST */
```

TEXT STATISTICS

```
  5168 characters
   170 lines
```

LEXICAL STATISTICS

```
    31 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     1 constants [character]
     3 constants [string]
    35 constants [numeric]
```

SYMBOL TABLE

```
    App_init        48     160
    App_next        74     161     165
    Degree          44      56     135
    Ldiff           45      61      62      85      86      99     100     101     123     127     144
    Loff            46      63      64      90      91     124     128     130
    NULL            78      79     143     162
    Pattern         41      55      98
    Plen            43      60      61      62      63      64      65      94     121+    135     138
          139
    Rdiff           45      62      63      67      84      85      87     102     113     123     125+
          127     135     139
    Roff            46      64      68      89      90      92     124     126+    128     130     138
    TEST           147
    Text            41      54      81      98     137
    Textloc         42      71      81      98     137
    a       76      99     100     106     107     108     109     113
    argc           149     154
    argv           149     160+
    atoi           160
    b       76     101     106     107
    begin          151     161     162     164+    165
    c       76     102     108     109
    degree          48      56
    end             74     137     138     151     161     164     165
    exit           157
    free           144
    h       34      35      36
    howclose        74     139     152     161     164     165
    i       50      65+     67+     68      76      94+     98      99     100     101     102     113
          121+    123+    124+    125+    126+    127+    128+    130+
    main           149
    malloc          61
    pattern         48      55      60
    printf         164
    puts           156
    start           74      78      79     138     143
    stdio           34
    stdlib          35
    string          36
    strlen          60
    temp            76      84      86      89      91
    text            48      54
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** arccrc16.c -- calculate 16-bit CRC for file(s)
** rev. Feb. 1992
** public domain by Raymond Gardner
**                  Englewood, Colorado
**
** This program uses the same CRC calculation as ARC (SEA) and LHA (Yoshi)
*/

#include <stdio.h>
#include "crc.h"

#define bufsiz (16*1024)

static WORD crc_table[256];

/*
** I determined this function empirically, by examining the
** table for patterns, and programming via trial-and-error.
** Don't ask me how it works or if it can be generalized for
** other CRC polynomials.
*/

void init_crc_table(void)
{
      int i, j;
      WORD k;

      for (i = 0; i < 256; i++)
      {
            k = 0xC0C1;
            for (j = 1; j < 256; j <<= 1)
            {
                  if (i & j)
                        crc_table[i] ^= k;
                  k = (k << 1) ^ 0x4003;
            }
      }
}

/*
** crc_calc() -- calculate cumulative crc-16 for buffer
*/

WORD crc_calc(WORD crc, char *buf, unsigned nbytes)
{
      unsigned char *p, *lim;

      p = (unsigned char *)buf;
      lim = p + nbytes;
      while (p < lim)
      {
            crc = (crc >> 8 ) ^ crc_table[(crc & 0xFF) ^ *p++];
      }
      return crc;
}

void do_file(char *fn)
{
      static char buf[bufsiz];
      FILE *f;
      int k;
      WORD crc;

      f = fopen(fn, "rb");
      if (f == NULL)
      {
            printf("%s: can't open file\n", fn);
            return;
      }
      crc = 0;
      while ((k = fread(buf, 1, bufsiz, f)) != 0)
            crc = crc_calc(crc, buf, k);
      fclose(f);
      printf("%-14s %04X\n", fn, crc);
}

#ifdef TEST

int main(int argc, char **argv)
{
      int i;

      if (argc < 2)
      {
            fprintf(stderr, "Usage: crc filename [filename...]\n");
            return EXIT_FAILURE;
      }
      init_crc_table();
      for (i = 1; i < argc; i++)
            do_file(argv[i]);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2050 characters
    97 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     5 constants [string]
    18 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            89
    EXIT_SUCCESS            94
    FILE            63
    NULL            68
    TEST            80
    WORD            17      29      47+     65
    argc            82      86      92
    argv            82      93
    buf             47      51      62      74      75
    bufsiz          15      62      74
    crc             47      55+     57      65      73      75+     77
    crc_calc        47      75
    crc_table       17      37      55
    do_file         60      93
    f       63      67      68      74      76
    fclose          76
    fn              60      67      70      77
    fopen           67
    fprintf         88
    fread           74
    h       12
    i       28      31+     36      37      84      92+     93
    init_crc_table          26      91
    j       28      34+     36
    k       29      33      37      38+     64      74      75
    lim             49      52      53
    main            82
    nbytes          47      52
    p       49      51      52      53      55
    printf          70      77
    stderr          88
    stdio           12
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Header for conveniently dealing with ASCII control characters
  5 | */
  6 |
  7 | #undef NUL
  8 | #undef SOH
  9 | #undef STX
 10 | #undef ETX
 11 | #undef EOT
 12 | #undef ENQ
 13 | #undef ACK
 14 | #undef BEL
 15 | #undef BS_
 16 | #undef HT_
 17 | #undef LF_
 18 | #undef VT_
 19 | #undef FF_
 20 | #undef CR_
 21 | #undef SO_
 22 | #undef SI_
 23 | #undef DLE
 24 | #undef DC1
 25 | #undef DC2
 26 | #undef DC3
 27 | #undef DC4
 28 | #undef NAK
 29 | #undef SYN
 30 | #undef ETB
 31 | #undef CAN
 32 | #undef EM_
 33 | #undef SUB
 34 | #undef ESC
 35 | #undef FS_
 36 | #undef GS_
 37 | #undef RS_
 38 | #undef US_
 39 | #undef DEL
 40 |
 41 | enum Cchar_T { NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
 42 |                BS_, HT_, LF_, VT_, FF_, CR_, SO_, SI_,
 43 |                DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
 44 |                CAN, EM_, SUB, ESC, FS_, GS_, RS_, US_,
 45 |                DEL = '\x7f' };
```

TEXT STATISTICS

```
   775 characters
    45 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
    33 preprocessor instructions
     1 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

ACK	13	41
BEL	14	41
BS_	15	42
CAN	31	44
CR_	20	42
Cchar_T	41	
DC1	24	43
DC2	25	43
DC3	26	43
DC4	27	43
DEL	39	45
DLE	23	43
EM_	32	44
ENQ	12	41
EOT	11	41
ESC	34	44
ETB	30	43
ETX	10	41
FF_	19	42
FS_	35	44
GS_	36	44
HT_	16	42
LF_	17	42
NAK	28	43
NUL	7	41
RS_	37	44
SI_	22	42
SOH	8	41
SO_	21	42
STX	9	41
SUB	33	44
SYN	29	43
US_	38	44
VT_	18	42

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ASSIGNPR.C
**
**  Multiple printer support with default to a single printer
**  connected to the PRN device.
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include "sniprint.h"

FILE *printer[NUM_OF_PRNTRS] = {stdprn};

/*
**  assign_printer()
**
**  Call with printer number and device name
**
**     printer number should be in the range of 0 to NUM_OF_PRNTRS-1
**     device should be "LPT1", "LPT2", "LPT3", "COM1", COM2", or a log file
**
**  Returns 0 if successful, -1 if error
**
**  Then do all printer output with fprintf(), fputs(), fputc(), etc.
**  using printer[printer_number] as the output stream
*/

int assign_printer(int number, char *device)
{
      FILE *fp;

      if (NUM_OF_PRNTRS <= number || NULL == (fp = fopen(device, "w")))
            return -1;
      printer[number] = fp;
      return 0;
}

#ifdef TEST                                    /* Test code follows    */

main()
{                                        /* Leave printer[0] = stdprn    */
      assign_printer(1, "CON");          /* Set printer[1] to the screen */
      assign_printer(2, "p.log");        /* Set printer[2] to log file   */
      fputs("This is a printer test\n", printer[0]);
      fputs("This is a screen test\n", printer[1]);
      fputs("This is a log test\n", printer[2]);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1686 characters
    57 lines

LEXICAL STATISTICS

     8 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     7 constants [string]
     8 constants [numeric]

SYMBOL TABLE

    FILE            19      37
    NULL            39
    NUM_OF_PRNTRS           19      39
    TEST            45
    assign_printer          35      49      50
    device          35      39
    fopen           39
    fp              37      39      41
    fputs           51      52      53
    main            47
    number          35      39      41
    printer         19      41      51      52      53
    stdprn          19
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Form a command string for ANSI.SYS to set a given video attribute
**
**  Public domain demo by Bob Stout
*/

#include <string.h>

#include "scrnmacs.h"

static void add_str(char *, char *);

/*
**  Example:
**   Video attribute of yellow text on blue background = BG_(BLUE_)+YELLOW_
*/

char *make_ansi(int vatr)
{
        static char string[40];

        static char *fore[8] = {"30","34","32","36","31","35","33","37"};
        static char *back[8] = {"40","44","42","46","41","45","43","47"};

        strcpy(string, "\033[");
        if (vatr == 0x07)
                strcat(string, "0");
        else
        {
                if (vatr & 0x80)
                        add_str(string, "5");
                if (vatr & 0x08)
                        add_str(string, "1");
                add_str(string, fore[vatr & 0x07]);
                add_str(string, back[(vatr & 0x70) >> 4]);
        }
        strcat(string, "m");
        return string;
}

static void add_str(char *string1, char *string2)
{
        char last_char;

        last_char = string1[strlen(string1) - 1];
        if (last_char != '[')
                strcat(string1, ";");
        strcat(string1, string2);
}
```

TEXT STATISTICS

```
  1281 characters
    51 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     1 constants [character]
    23 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

add_str	13	33	35	36	37	43				
back	25	37								
fore	24	36								
h 9										
last_char	45	47	48							
make_ansi	20									
strcat	29	39	49	50						
strcpy	27									
string	9	22	27	29	33	35	36	37	39	40
string1	43	47+	49	50						
string2	43	50								
strlen	47									
vatr	20	28	32	34	36	37				

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)BACSTD.h
*
* @(#)Headerfile for the BACSTD library.
*
* This library is avalable for the following memorymodels:
* Small (bacstd_S.lib), Medium (bacstd_M.lib), Compact (bacstd_C.lib),
* Large (bacstd_L.lib) and Huge (bacstd_H.lib)
*
* BACSTD.lib contains several basic routines for BORLAND C and TURBO C
*
* For further details see the textfile: BACSTD.TXT
*
*----------------------------------------------------------------------------
*
*****************************************************************************
*@(#)1991-1995 Erik Bachmann (E-mail: ebp@dde.dk)
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <stdio.h>                      /* FILE */
#include <stdarg.h>                     /* va_list */
#include "modulinf.h"

/*** DEFINITIONS ************************************/
#define _cdecl cdecl
                                        /* Library => EXTERN */
#if !defined(BACSTD_LIB_H)
#  define   BACSTD_LIB_H


#  if defined(__TINY__) || defined(__SMALL__)
#     define _CfnTYPE near              /* neardef for functions */
#     define _CdtTYPE near              /* neardef for data */
#  else
#     define _CfnTYPE far               /* fardef for functions */
#     define _CdtTYPE far               /* fardef for data */
#  endif


/*** GLOBAL DATA ************************************/
   enum BOOLEAN   { FALSE, TRUE };

   #define  complement(base)  ~base + 1

#endif


/*** PROTOTYPES *************************************/

/* E_HANDL.C */
int handler ( int errval , int ax , int bp , int si );

/* STRCASE.C */
unsigned char _CfnTYPE *strcase ( unsigned char *pszStr ,
         unsigned char *pszOrder );

/* STRREPC.C */
int _CfnTYPE strrepc ( char *pszStr , char cFrom , char cTo );

/* MODULUS.C */
int _CfnTYPE modulus10 ( char *pszBase );
int _CfnTYPE modulus11 ( char *pszBase );
int _CfnTYPE check_modulus10 ( char *pszBase );
int _CfnTYPE check_modulus11 ( char *pszBase );

/* REPSTR.C */
int _CfnTYPE repstr ( FILE *fpIn , FILE *fpOut , char *PatternTable [][2 ]);

/* CMPSTR.C */
int far cmpstr ( unsigned char *pszStr1 , unsigned char *pszStr2 ,
         unsigned char *pszOrder );
int far tcmpstr ( unsigned char *pszStr1 , unsigned char *pszStr2 ,
         unsigned char *pszOrder , unsigned char *pszMask ,
         unsigned char *pszTrunc );
void state ( int iOrder );

/* STRNSUB.C */
char _CfnTYPE *strnsub ( char *pszString , char *pszPattern ,
         char *pszReplacement , int iMaxLength );

/* TIME__.C */
char _CfnTYPE *time_mac_conv ( char *pszTime );

/* DATE__.C */
char far *date_mac_conv ( char *pszDate );

/* STRTRIM.C */
int _CfnTYPE strtrimr ( char *pszStr );
int _CfnTYPE strtriml ( char *pszStr );
int _CfnTYPE strtrim ( char *pszStr );

/* STRTRIMC.C */
int _CfnTYPE strtrimcr ( char *szStr , char *szSet );
int _CfnTYPE strtrimcl ( char *szStr , char *szSet );
int _CfnTYPE strtrimc ( char *szStr , char *szSet );
int _CfnTYPE rep_last_char ( char *pszStr , char cChar1 , char cChar2 );
```

TEXT STATISTICS

```
  3281 characters
   100 lines
```

LEXICAL STATISTICS

```
    23 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    BACSTD_LIB_H            31      32
    BOOLEAN         45
    FALSE           45
    FILE            71+
    PatternTable            71
    TRUE            45
    _CdtTYPE        37      40
    _CfnTYPE        36      39      58      62      65      66      67      68      71      82      86
            92      93      94      97      98      99     100
    __SMALL__       35
    __TINY__        35
    _cdecl          29
    ax              55
    base            47+
    bp              55
    cChar1         100
    cChar2         100
    cFrom           62
    cTo             62
    cdecl           29
    check_modulus10         67
    check_modulus11         68
    cmpstr          74
    complement              47
    date_mac_conv           89
    define          32      36      37      39      40
    defined         31      35+
    endif           41
    errval          55
    far             39      40      74      76      89
    fpIn            71
    fpOut           71
    h       24      25
    handler         55
    iMaxLength              83
    iOrder          79
    modulus10       65
    modulus11       66
    near            36      37
    pszBase         65      66      67      68
    pszDate         89
    pszMask         77
    pszOrder        59      75      77
    pszPattern              82
    pszReplacement          83
    pszStr          58      62      92      93      94     100
    pszStr1         74      76
    pszStr2         74      76
    pszString       82
    pszTime         86
    pszTrunc        78
    rep_last_char          100
    repstr          71
    si              55
    state           79
    stdarg          25
    stdio           24
    strcase         58
    strnsub         82
    strrepc         62
    strtrim         94
    strtrimc        99
    strtrimcl       98
    strtrimcr       97
    strtriml        93
    strtrimr        92
    szSet           97      98      99
    szStr           97      98      99
    tcmpstr         76
    time_mac_conv           86
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BASCNVRT.C - Convert between number bases
**
**  public domain demo by Bob Stout
*/

#include <stdlib.h>
#include "extkword.h"
#include "numcnvrt.h"

/*
**  Calling parameters: 1 - Number string to be converted
**                      2 - Buffer for the  converted output
**                      3 - Radix (base) of the input
**                      4 - Radix of the output
**
**  Returns: Pointer to converted output
**
**  Uses LTOSTR.C from SNIPPETS
*/

char *base_convert(const char *in, char *out, size_t len, int rin, int rout)
{
      long n;
      char *dummy;

      n = strtol(in, &dummy, rin);
      return ltostr(n, out, len, rout);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      int rin, rout;
      char buf[40];

      if (4 > argc)
      {
            puts("Usage: BASCNVRT <number> <base_in> <base_out>");
            return(-1);
      }
      rin  = atoi(argv[2]);
      rout = atoi(argv[3]);
      printf("%s (base %d) = %s (base %d)\n", argv[1], rin,
            base_convert((const char *)argv[1], buf, 40, rin, rout), rout);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1238 characters
    55 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     4 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    TEST            33
    argc            38      43
    argv            38      48      49      50      51
    atoi            48      49
    base_convert            24      51
    buf             41      51
    dummy           27      29
    h         9     35      36
    in              24      29
    len             24      30
    ltostr          30
    main            38
    n       26      29      30
    out             24      30
    printf          50
    puts            45
    rin             24      29      40      48      50      51
    rout            24      30      40      49      51+
    size_t          24
    stdio           35
    stdlib           9      36
    strtol          29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BASIC-like string operations
**
**  public domain by Bob Stout
*/

#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <limits.h>
#include <assert.h>
#include "sniptype.h"
#include "bastrngs.h"

static int stralloc_ptr;
static char *strings[8];
static int str_tag[8];

/*
**  stralloc() is the key function in this package, maintaining a pool of
**  reusable strings.
*/

char *stralloc(size_t length)
{
      register int i;

      if (UINT_MAX == length)       /* Assume size_t == unsigned int    */
            return NULL;

      i = stralloc_ptr++;
      ++length;                     /* Allow for terminating NUL        */

      if ((!strings[i]) || (length > strlen(strings[i])))
      {
            strings[i] = (char *)realloc(strings[i], length);
            assert(NULL != strings[i]);
            str_tag[i] = -1;
      }
      else  str_tag[i] = 0;
      stralloc_ptr &= 7;
      return (strings[i]);
      /* Maintains 8 strings in a circular buffer */
}

/*
**  free the string pool.
*/

void str_free(char *string)
{
      register int i;

      for (i = 0; i < 8; ++i)
      {
            if (strings[i] == string)
            {
                  if (str_tag[i])
                        free(strings[i]);
                  return;
            }
      }
}

/*
**  return the leftmost N characters from a string, equivalent to LEFT$
*/

char *left(char *string, size_t N)
{
      char *buf;
      size_t strlength = strlen(string);

      if (N > strlength)
            N = strlength;
      buf = stralloc(N);
      memcpy(buf, string, N);
      buf[N] = NUL;
      return buf;
}

/*
**  return the rightmost N characters from a string, equivalent to RIGHT$
*/

char *right(char *string, size_t N)
{
      char *buf;
      size_t strlength = strlen(string);

      if (N > strlength)
            N = strlength;
      buf = stralloc(N);
      strcpy(buf, &string[strlength-N]);
      return buf;
}

/*
```

```
101 | **  return a substring, N characters long beginning at position M,
102 | **  equivalent to MID$
103 | */
104 |
105 | char *mid(char *string, size_t M, size_t N)
106 | {
107 |       char *buf;
108 |       size_t strlength = strlen(string);
109 |
110 |       if (M > strlength)
111 |             return NULL;
112 |       if (N > (strlength - M))
113 |             N = strlength - M;
114 |       buf = stralloc(N);
115 |       memcpy(buf, &string[M-1], N);
116 |       buf[N] = NUL;
117 |       return buf;
118 | }
119 |
120 | /*
121 | **  string concatenation function, equivalent to A$=B$+C$+...
122 | */
123 |
124 | char *string_add(char *string, ...)
125 | {
126 |       va_list arg_ptr;
127 |       char *temp1, *temp2, *buf;
128 |
129 |       va_start(arg_ptr, string);
130 |       temp1 = string;
131 |       do
132 |       {
133 |             if(NULL == (temp2 = va_arg(arg_ptr, char *)))
134 |                   break;
135 |             buf = stralloc(strlen(temp1) + strlen(temp2));
136 |             temp1 = strcat(strcpy(buf, temp1), temp2);
137 |       } while (NULL != temp2);
138 |       return temp1;
139 | }
140 |
141 | /*
142 | **  create a string of repeated characters, equivalent to STRING$()
143 | */
144 |
145 | char *string(int ch, size_t N)
146 | {
147 |       char *buf;
148 |
149 |       if (N)
150 |       {
151 |             buf = stralloc(N);
152 |             memset(buf, ch, N);
153 |       }
154 |       buf[N] = NUL;
155 |       return buf;
156 |
157 | }
158 |
159 | /*
160 | **  BASIC INSTR$() work alike - returns position (1-based) of findstr in
161 | **  string, starting search at position start_pos (also 1-based).
162 | **
163 | **  Function suggested by Tika Carr
164 | */
165 |
166 | unsigned int instr(const unsigned int start_pos,
167 |                    const char        *string,
168 |                    const char        *findstr)
169 | {
170 |       char *p;
171 |       unsigned int pos;
172 |
173 |       if (start_pos > strlen(string))
174 |             return 0;
175 |       else  pos = start_pos - 1;    /* BASIC offsets are one-based, not
176 |                                        zero-based as in C               */
177 |
178 |       if (NULL != (p = strstr(string + pos, findstr)))
179 |             return (int)(p - (char *)string) + 1; /* Position 0 = position 1 */
180 |       else  return 0;
181 | }
182 |
183 | #ifdef TEST
184 |
185 | /*
186 | **  Demo main()
187 | */
188 |
189 | main()
190 | {
191 |       char *x = "European", *y = "Hardware", *z = "Skaters", *q;
192 |       char *a = "This is a test", *b = "is" ,
193 |            *c = "\nSearching for \"%s\" in \"%s\" starting at position %d\n";
194 |       unsigned int i, pos;
195 |
196 |       z = string_add(left(x, 2), right(y, 2), mid(z, 2, 2), "!", NULL);
197 |       q = string('!', 4);
198 |       printf("%s%s\n", z, q);
199 |
200 |       for (i = 2; i < strlen(a); i+= 2)
```

```
 201 |       {
 202 |             printf(c, b, a, i);
 203 |             if (0 != (pos = instr(i, a, b)))
 204 |                   printf("Found at position %d\n", pos);
 205 |             else  puts("Not found");
 206 |       }
 207 |
 208 |       return 0;
 209 | }
 210 |
 211 | #endif /* TEST */
```

TEXT STATISTICS

```
  4764 characters
   211 lines
```

LEXICAL STATISTICS

```
    17 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     1 constants [character]
    12 constants [string]
    21 constants [numeric]
```

SYMBOL TABLE

```
    M      105     110     112     113     115
    N       71      76      77      78      79      80      88      93      94      95      96     105
           112     113     114     115     116     145     149     151     152     154
    NUL             80     116     154
    NULL            31      39     111     133     137     178     196
    TEST           183
    UINT_MAX        30
    a      192     200     202     203
    arg_ptr        126     129     133
    assert          13      39
    b      192     202     203
    buf             73      78      79      80      81      90      95      96      97     107     114
           115     116     117     127     135     136     147     151     152     154     155
    c      193     202
    ch             145     152
    findstr        168     178
    free            61
    h        9      10      11      12      13
    i       28      33      36+     38+     39      40      42      44      54      56+     58      60
           61     194     200+    202     203
    instr          166     203
    left            71     196
    length          26      30      34      36      38
    limits          12
    main           189
    memcpy          79     115
    memset         152
    mid            105     196
    p      170     178     179
    pos            171     175     178     194     203     204
    printf         198     202     204
    puts           205
    q      191     197     198
    realloc         38
    right           88     196
    size_t          26      71      74      88      91     105+    108     145
    start_pos      166     173     175
    stdarg          11
    stdio            9
    str_free        52
    str_tag         19      40      42      60
    stralloc        26      78      95     114     135     151
    stralloc_ptr            17      33      43
    strcat         136
    strcpy          96     136
    string          10      52      58      71      74      79      88      91      96     105     108
           115     124     129     130     145     167     173     178     179     197
    string_add             124     196
    strings         18      36+     38+     39      44      58      61
    strlen          36      74      91     108     135+    173     200
    strlength       74      76      77      91      93      94      96     108     110     112     113
    strstr         178
    temp1          127     130     135     136+    138
    temp2          127     133     135     136     137
    va_arg         133
    va_list        126
    va_start       129
    x      191     196
    y      191     196
    z      191     196+    198
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  BASIC-like string operations
  5 | */
  6 |
  7 | #ifndef BASTRNGS__H
  8 | #define BASTRNGS__H
  9 |
 10 | #include <stdlib.h>                             /* For size_t     */
 11 |
 12 | char *stralloc(size_t length);
 13 | void  str_free(char *string);
 14 | char *left(char *string, size_t N);
 15 | char *right(char *string, size_t N);
 16 | char *mid(char *string, size_t M, size_t N);
 17 | char *string_add(char *string, ...);
 18 | char *string(int ch, size_t N);
 19 |
 20 | unsigned int instr(const unsigned int start_pos,
 21 |                    const char        *string,
 22 |                    const char        *findstr);
 23 |
 24 | #endif /* BASTRNGS__H */
```

TEXT STATISTICS

```
   635 characters
    24 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    BASTRNGS__H              7       8
    M       16
    N       14      15      16      18
    ch              18
    findstr         22
    h       10
    instr           20
    left            14
    length          12
    mid             16
    right           15
    size_t          12      14      15      16+     18
    start_pos       20
    stdlib          10
    str_free        13
    stralloc        12
    string          13      14      15      16      17      18      21
    string_add              17
```

```
/* +++Date last modified: 05-Jul-1997 */

/*FILE*************************************************************/
/* File Id:                    bcd.c.                               */
/* Author:                     Stan Milam.                          */
/* Date Written:               31-Jan-95.                           */
/* Description:                                                     */
/*     Routines to manage binary coded decimal.                     */
/*                                                                  */
/*     bcd_to_double() - Convert BCD to type double value.          */
/*     double_to_bcd() - Convert double type value to BCD.          */
/*                                                                  */
/* Placed in the public domain by the author, 8-Sep-95              */
/*                                                                  */
/*****************************************************************FILE*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <float.h>
#include <math.h>
#include "snipmath.h"

/*FUNCTION*********************************************************/
/* Name: bcd_to_double().                                           */
/*                                                                  */
/* Description:                                                     */
/*     This function will convert buffer containing binary coded dec- */
/*     imal to a double.                                            */
/*                                                                  */
/* Arguments:                                                       */
/*     void     *buf  - Address of buffer containing binary coded  */
/*                      decimal number.                             */
/*     size_t   buflen- Length of the buffer.                       */
/*     int      digits- The number of digits to the right of the    */
/*                      decimal point.                              */
/*                                                                  */
/* Return Value:                                                    */
/*     A value of type double which should be the equivalent of the */
/*     BCD value.                                                   */
/*                                                                  */
/*****************************************************************/

double bcd_to_double(void *buf, size_t len, int digits)
{
      double   rv = 0.0;
      char     *buffer = (char *) buf;
      size_t   high, low, index, max = len - 1;

      digits = abs(digits);

    /*****************************************************************/
    /* Loop through the buffer repeatedly extracting the high and low */
    /* 4 bits from each byte and calculating the return value.       */
    /*****************************************************************/

      for (index = 0; index < max; index++)
      {
            low  = buffer[index] & 0x0f;
            high = (buffer[index] & 0xf0) >> 4;
            rv = (( rv * 10.0 + high ) * 10.0 + low);
      }

    /*****************************************************************/
    /* The first byte of the buffer contains the lowest order digit  */
    /* in the upper 4 bits and the sign in the lower 4 bits.         */
    /*****************************************************************/

      low  = buffer[max] & 0x0f;
      high = (buffer[max] & 0xf0) >> 4;
      rv = rv * 10.0 + high;
      if (digits > 0)
            rv /= pow(10, digits);
      if (low == 0x0d)
            rv = -rv;
      return rv;
}

/*FUNCTION*********************************************************/
/* Name: double_to_bcd().                                           */
/*                                                                  */
/* Description:                                                     */
/*     This function will convert a value of type double to a legit- */
/*     mate Binary Coded Decimal (BCD) value.                       */
/*                                                                  */
/* Arguments:                                                       */
/*     double   arg    - The value to be converted.                 */
/*     char     *buf   - The buffer where the BCD value is stored.  */
/*     size_t   length - The number of significant digits to store in */
/*                       BCD buffer.                                */
/*     size_t   digits - The number of digits to the right of the   */
/*                       decimal point to be stored in the BCD      */
/*                       buffer.                                    */
/*                                                                  */
/* Return Value:                                                    */
/*     An integer value indicating the length of the BCD value in the */
/*     buffer.  -1 is returned if an error occured.                 */
/*                                                                  */
/*****************************************************************FUNCTION*/

```

```
101 | int double_to_bcd(double arg, char *buf, size_t length, size_t digits )
102 | {
103 |       char wrkbuf[50], format[DBL_DIG + 1];
104 |       int  y_sub, x_sub, rv, negative=0;
105 |
106 |     /*****************************************************************/
107 |     /* Do a couple of sanity checks first.                           */
108 |     /*****************************************************************/
109 |
110 |       if ((length == 0 && digits == 0) || (length + digits > DBL_DIG))
111 |             rv = -1;
112 |       else
113 |       {
114 |         /*************************************************************/
115 |         /* If the double argument is negative make a note of it and   */
116 |         /* con- vert the value to be positive.                        */
117 |         /*************************************************************/
118 |
119 |             if (arg < 0.0)
120 |             {
121 |                   arg = -arg;
122 |                   negative = 1;
123 |             }
124 |
125 |         /*************************************************************/
126 |         /* Adjust for decimal digits.                                 */
127 |         /*************************************************************/
128 |
129 |             if (digits > 0)
130 |             {
131 |                   length += digits;
132 |                   arg *= pow( 10, digits );
133 |             }
134 |
135 |         /*************************************************************/
136 |         /* Build the format string, build the string and compute the  */
137 |         /* return value.                                              */
138 |         /*************************************************************/
139 |
140 |             sprintf( format, "%%0%d.0f", length );
141 |             sprintf( wrkbuf, format, floor( arg ) );
142 |             if ((rv = (length / 2 ) + (length / 2 != 0)) == 0)
143 |                   rv = 1;
144 |
145 |         /*************************************************************/
146 |         /* Compute the subscript values and clear the BCD buffer.     */
147 |         /*************************************************************/
148 |
149 |             y_sub = rv - 1;
150 |             x_sub = strlen( wrkbuf ) - 1;
151 |             memset( buf, 0, y_sub + 1 );
152 |
153 |         /*************************************************************/
154 |         /* Plug in the sign bits and first BCD digit.                 */
155 |         /*************************************************************/
156 |
157 |             buf[y_sub] = negative == 1 ? 0xd : 0xc;
158 |             buf[y_sub--] |= ( ( wrkbuf[x_sub--] - '0' ) << 4 );
159 |
160 |         /*************************************************************/
161 |         /* While we have more digits to plug....                      */
162 |         /*************************************************************/
163 |
164 |             while ( --length > 0 )
165 |             {
166 |
167 |             /*********************************************************/
168 |             /* Do the low nibble of the BCD byte.                      */
169 |             /*********************************************************/
170 |
171 |                   buf[y_sub] = wrkbuf[x_sub--] - '0';
172 |                   if (--length <= 0)
173 |                         break;
174 |
175 |             /*********************************************************/
176 |             /* Now do the high nibble.                                 */
177 |             /*********************************************************/
178 |
179 |                   buf[y_sub--] |= ((wrkbuf[x_sub--] - '0') << 4);
180 |             }
181 |       }
182 |       return rv;
183 | }
184 | /**/
185 | #ifdef TEST
186 |
187 | typedef struct {
188 |     double value, expect;
189 |     int    length, digits;
190 | } TEST_T;
191 |
192 | int main(void)
193 | {
194 |       double rv, value;
195 |       char   wrkbf[25];
196 |       int    rc, x_sub, y_sub, size, len, digits;
197 |       char   format[] = "    %10.3f      %d          %d        %d      ";
198 |       TEST_T testvals[] = {
199 |             {  12345.67,     123.45,   5, 0 },
200 |             {  12345.67,   12345.0,   5, 1 },
```

```
201 |             {  12345.67,   12345.67,  4, 3 },
202 |             {  12345.678,   2345.67,  1, 2 },
203 |             { -12345.00,  -12345.00,  8, 2 },
204 |             { -1234.56,  -12345.00,   5, 3 },
205 |             {  1234.567,      0.60,   1, 3 }
206 |       };
207 |
208 |       size = sizeof( testvals ) / sizeof( TEST_T );
209 |       printf("    Double       Length     Digits    Return\n");
210 |       printf("    Argument     Argument   Argument  Value    Buffer\n");
211 |       printf("    ================================================="
212 |              "=========\n");
213 |
214 |       for (x_sub = 0; x_sub < size; x_sub++)
215 |       {
216 |             len = testvals[x_sub].length;
217 |             digits = testvals[x_sub].digits;
218 |             value  = testvals[x_sub].value;
219 |
220 |             rc = double_to_bcd(value, wrkbf, len, digits);
221 |             if (rc > 0)
222 |             {
223 |                   printf( format, value, len, digits, rc );
224 |                   for ( y_sub = 0; y_sub < rc; y_sub++ )
225 |                         printf("%02X ", wrkbf[y_sub]);
226 |                   printf("\n");
227 |             }
228 |       }
229 |       return 0;
230 | }
231 | #endif
```

TEXT STATISTICS

```
 10312 characters
   231 lines
```

LEXICAL STATISTICS

```
    92 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     3 constants [character]
     9 constants [string]
    74 constants [numeric]
```

WARNINGS

```
     line   184 : control character ignored
```

SYMBOL TABLE

```
    DBL_DIG         103     110
    TEST            185
    TEST_T          190     198     208
    abs              50
    arg             101     119     121+    132     141
    bcd_to_double            44
    buf              44      47     101     151     157     158     171     179
    buffer           47      59      60      69      70
    digits           44      50+     72      73     101     110+    129     131     132     189     196
           217+     220     223
    double_to_bcd            101     220
    expect          188
    floor           141
    format          103     140     141     197     223
    h        17      18      19      20      21
    high             48      60      61      70      71
    index            48      57+     59      60
    len              44      48     196     216     220     223
    length          101     110+    131     140     142+    164     172     189     216
    low              48      59      61      69      74
    main            192
    math             21
    max              48      57      69      70
    memset          151
    negative        104     122     157
    pow              73     132
    printf          209     210     211     223     225     226
    rc              196     220     221     223     224
    rv               46      61+     71+     73      75+     76     104     111     142     143     149
           182     194
    size            196     208     214
    size_t           44      48     101+
    sprintf         140     141
    stdio            17
    stdlib           18
    string           19
    strlen          150
    testvals        198     208     216     217     218
    value           188     194     218+    220     223
    wrkbf           195     220     225
    wrkbuf          103     141     150     158     171     179
    x_sub           104     150     158     171     179     196     214+    216     217     218
    y_sub           104     149     151     157     158     171     179     196     224+    225
```

```
/* +++Date last modified: 05-Jul-1997 */

/*-----------------01-10-95 02:38pm-----------------
--  BCD conversion routines, released to the public
--  domain by Jeff Dunlop
------------------------------------------------*/

#include "sniptype.h"         /* For True_, False_    */
#include "snipmath.h"

static int BCDLen = 8;

void SetBCDLen(int n)
{
      BCDLen = n;
}

long BCDtoLong(char *BCDNum)
{
      int i, high, low;
      Boolean_T neg;

      long total = 0;

      for (i = 0; i < BCDLen - 1; i++)
      {
            low    = 0x0f & BCDNum[i];
            high   = (0xf0 & BCDNum[i]) >> 4;
            total *= 10;
            total += high;
            total *= 10;
            total += low;
      }
      high   = (BCDNum[BCDLen - 1] & 0xf0) >> 4;
      low    = BCDNum[BCDLen - 1] & 0x0f;
      total *= 10;
      total += high;

      neg = (low == 0x0c) ? 1 : -1;

      return total * neg;
}

void LongtoBCD(long num, char BCDNum[])
{
      int i, high, low;
      Boolean_T neg = False_;

      if ( num < 0 )
      {
            neg = True_;
            num = 0 - num;
      }

      low  = neg ? 0x0d : 0x0c;
      high = (int)(num % 10);
      num /= 10;
      BCDNum[BCDLen - 1] = (char)((high << 4) | low);

      for (i = BCDLen - 2; i >= 0; i--)
      {
            low  = (int)(num % 10);
            num /= 10;
            high = (int)(num % 10);
            num /= 10;
            BCDNum[i] = (char)((high << 4) | low);
      }
}

#ifdef TEST

#include <stdio.h>

void show_BCD(char c[]);

int main(void)
{
      long a = 12345678L,
           b;
      char c[10];

      SetBCDLen(10);

      LongtoBCD(a, c);
      show_BCD(c);
      b= BCDtoLong(c);
      printf("Value is %ld\n", b);

      a = -a;
      LongtoBCD(a, c);
      show_BCD(c);
      b= BCDtoLong(c);
      printf("Value is %ld\n", b);

      return 0;
}

void show_BCD(char c[])
{
      int i;
```

```
 101 |
 102 |       printf("BCD: ");
 103 |       for (i = 0; i < 10; ++i)
 104 |       {
 105 |             printf("[%d]", c[i] >> 4);
 106 |             if (9 == i)
 107 |                   printf("<sign>");
 108 |             printf("[%d]", c[i] &0xf);
 109 |       }
 110 |       puts("");
 111 | }
 112 |
 113 | #endif /* TEST */
```

TEXT STATISTICS

```
  2256 characters
   113 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     9 constants [string]
    42 constants [numeric]
```

SYMBOL TABLE

```
    BCDLen          11      15      25      34      35      58      60
    BCDNum          18      27      28      34      35      44      58      66
    BCDtoLong       18      86      92
    Boolean_T       21      47
    False_          47
    LongtoBCD       44      84      90
    SetBCDLen       13      82
    TEST            70
    True_           51
    a       78      84      89+     90
    b       79      86      87      92      93
    c       74      80      84      85      86      90      91      92      98     105     108
    h       72
    high            20      28      30      34      37      46      56      58      64      66
    i       20      25+     27      28      46      60+     66     100     103+    105     106     108
    low             20      27      32      35      39      46      55      58      62      66
    main            76
    n       13      15
    neg             21      39      41      47      51      55
    num             44      49      52+     56      57      62      63      64      65
    printf          87      93     102     105     107     108
    puts           110
    show_BCD        74      85      91      98
    stdio           72
    total           23      29      30      31      32      36      37      41
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  Port Borland random() and randomize() functions to other compilers
   5 | */
   6 |
   7 | #ifndef BC_RAND__H
   8 | #define BC_RAND__H
   9 |
  10 | #include <stdlib.h>           /* For RAND_MAX, NULL, rand() and srand() */
  11 | #include <time.h>             /* For time()                             */
  12 |
  13 | #ifndef __TURBOC__
  14 |  #define random(num) (int)(((rand())*(long)(num))/(((long)RAND_MAX)+1))
  15 |  #define randomize() srand((unsigned)time(NULL)|1)
  16 | #endif
  17 |
  18 | #endif /* BC_RAND__H */
```

TEXT STATISTICS

```
   502 characters
    18 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     0 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    BC_RAND__H               7       8
    NULL            15
    RAND_MAX        14
    __TURBOC__              13
    h       10      11
    num             14+
    rand            14
    random          14
    randomize       15
    srand           15
    stdlib          10
    time            11      15
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  BHAMDEMO.C
  5 | **
  6 | **  Demonstration of Bresenham algorithms for line and circle.
  7 | **  public domain by Kurt Kuzba
  8 | */
  9 |
 10 | #include <stdio.h>
 11 | #include <stdlib.h>
 12 | #include <conio.h>
 13 |
 14 | #include "bresnham.h"
 15 |
 16 | int main()
 17 | {
 18 |       unsigned d = 0, c = 0, color = 0, lines[473][2];
 19 |
 20 |       srand(*((unsigned*)0x46c));
 21 |       setmode(0x13);    /* set to color graphics mode, clearing screen  */
 22 |       bresenham_circle(160, 100, 75, 15);
 23 |       for (d = 0; d < 64000l; d++)
 24 |       {
 25 |             if(*((char far*)0xa0000000l + d) == 15)
 26 |                   lines[c][1] = d / 320, lines[c++][0] = d % 320;
 27 |       }
 28 |       while (*((char far*)0x41a) == *((char far*)0x41c))
 29 |       {
 30 |             bresenham_circle(160, 100,
 31 |                   (++color & 127) + 80, (color / 5 ) & 255);
 32 |             d = rand() % 465 + 4;
 33 |             c = (d + 3) % 472;
 34 |             bresenham_line(lines[d][0], lines[d][1],
 35 |                   lines[c][0], lines[c][1], color & 255);
 36 |       }
 37 |       setmode(0x03);    /* set to color text mode, clearing screen      */
 38 |       getch();
 39 |       return 0;
 40 | }
```

TEXT STATISTICS

```
  1130 characters
    40 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
    38 constants [numeric]
```

SYMBOL TABLE

```
    bresenham_circle        22      30
    bresenham_line          34
    c       18      26+     33      35+
    color           18      31+     35
    conio           12
    d       18      23+     25      26+     32      33      34+
    far             25      28+
    getch           38
    h       10      11      12
    lines           18      26+     34+     35+
    main            16
    rand            32
    setmode         21      37
    srand           20
    stdio           10
    stdlib          11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  bigfac.c -- put into the public domain by Carl Declerck
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFLEN 8192
#define BUFFER ((char *) malloc(BUFFLEN))

int  main (void);
void multiply (char *, char *, char *);
void zero_buffer (char *);
void minus_one (char *);
int  isnull (char *);
void factorial (char *);

main (void)
{
      char *g = BUFFER;

      printf ("Enter a number: ");
      scanf ("%s", g);
      printf ("Factorial of %s is: ", g);
      factorial (g);
      printf ("%s\n", g);
      free (g);
      return 0;
}

void multiply (char *g1, char *g2, char *g3)
{
      int gp1, gp2, cumpos, respos, mod, div;
      int cmod, cdiv, resoff, wdig1, wdig2, base;

      zero_buffer (g3);
      for (gp2 = strlen(g2) - 1; gp2 >= 0; gp2--)
      {
            wdig2 = *(g2 + gp2) - 48;
            resoff = strlen(g2) - gp2 - 1;
            respos = BUFFLEN - resoff - 2;
            for (gp1 = strlen(g1) - 1; gp1 >= 0; gp1--)
            {
                  wdig1 = *(g1 + gp1) - 48;
                  mod = (wdig1 * wdig2) % 10;
                  div = (wdig1 * wdig2) / 10;
                  base = *(g3 + respos) - 48;
                  cmod = (base + mod) % 10;
                  cdiv = (base + mod) / 10 + div;
                  *(g3 + respos) = (char)(cmod + 48);
                  cumpos = --respos;
                  while (cdiv > 0)
                  {
                        base = *(g3 + cumpos) - 48;
                        *(g3 + cumpos--) = (char)((base + cdiv) % 10 + 48);
                        cdiv = (base + cdiv) / 10;
                  }
            }
      }
      for (respos = 0; *(g3 + respos) == '0'; respos++)
            ;
      strcpy (g3, (char *) (g3 + respos));
      if (*g3 == 0)
            strcpy (g3, "0");
}

void zero_buffer (char *buff)
{
      int cnt;

      for (cnt= 0; cnt < BUFFLEN; cnt++)
            *(buff + cnt) = '0';
      *(buff + BUFFLEN - 1) = 0;
}

void minus_one (char *g)
{
      int p;
      char digit;

      p = strlen(g) - 1;
      digit = *(g + p);
      while (digit == '0')
      {
            *(g + p--) = '9';
            digit = *(g + p);
      }
      *(g + p) -= 1;
}

int isnull (char *g)
{
      int p, ok = 1;

      for (p = 0; p < (int)(strlen(g)); p++)
            if (*(g + p) != '0')
                  ok = 0;
```

```
101 |       return (ok);
102 | }
103 |
104 | void factorial (char *g)
105 | {
106 |       char *h1 = BUFFER, *h2 = BUFFER;
107 |
108 |       strcpy (h1, "1");
109 |       while (!isnull(g))
110 |       {
111 |             multiply (h1, g, h2);
112 |             strcpy (h1, h2);
113 |             minus_one (g);
114 |       }
115 |       strcpy (g, h1);
116 |       free (h1);
117 |       free (h2);
118 | }
119 |
120 | /*
121 | **  The principal function is multiply(), it 'multiplies' two
122 | **  character-strings of arbitrary length and puts the result
123 | **  into a third.  8192 bytes is enough for 1000!, beyond that
124 | **  the buffer-size may need to be incremented.
125 | */
```

TEXT STATISTICS

```
  3052 characters
   125 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     5 constants [character]
     6 constants [string]
    31 constants [numeric]
```

SYMBOL TABLE

```
    BUFFER          12      23     106+
    BUFFLEN         11      12      44      74      76
    base            37      50      51      52      57      58      59
    buff            70      75      76
    cdiv            37      52      55      58      59+
    cmod            37      51      53
    cnt             72      74+     75
    cumpos          36      54      57      58
    digit           82      85      86      89
    div             36      49      52
    factorial       19      28     104
    free            30     116     117
    g       23      26      27      28      29      30      79      84      85      88      89      91
            94      98      99     104     109     111     113     115
    g1              34      45      47
    g2              34      40      42      43
    g3              34      39      50      53      57      58      63      65+     66      67
    gp1             36      45+     47
    gp2             36      40+     42      43
    h        7       8       9
    h1             106     108     111     112     115     116
    h2             106     111     112     117
    isnull          18      94     109
    main            14      21
    malloc          12
    minus_one       17      79     113
    mod             36      48      51      52
    multiply        15      34     111
    ok              96     100     101
    p       81      84      85      88      89      91      96      98+     99
    printf          25      27      29
    resoff          37      43      44
    respos          36      44      50      53      54      63+     65
    scanf           26
    stdio            7
    stdlib           8
    strcpy          65      67     108     112     115
    string           9
    strlen          40      43      45      84      98
    wdig1           37      47      48      49
    wdig2           37      42      48      49
    zero_buffer             16      39      70
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* BIGNUM.H
  4 | **
  5 | ** Header file with definition of BigNum type for Big Number Arithmetic.
  6 | **
  7 | ** Released to the public domain by Clifford Rhodes on June 15, 1995 with
  8 | ** no guarantees of any sort as to accuracy or fitness of use.
  9 | */
 10 |
 11 | /* Each int in the Big Number array will hold a digit up to MODULUS - 1.
 12 | ** Choosing 10000 makes testing easy because each digit contains 4 decimal
 13 | ** places.
 14 | */
 15 |
 16 | #ifndef BIGNUM__H
 17 | #define BIGNUM__H
 18 |
 19 | #include "minmax.h"
 20 |
 21 | #define MODULUS  10000    /* Warning: Must be <= USHRT_MAX! */
 22 |
 23 | typedef unsigned short UShort;
 24 | typedef unsigned long  ULong;
 25 |
 26 | /* The Big Number is contained in a structure that has a length, nlen, and
 27 | ** an array, n[], of unsigned shorts to hold the 'digits'. The most
 28 | ** significant digit of the big number is at n[0]. The least significant
 29 | ** digit is at n[nlen - 1];
 30 | */
 31 |
 32 | typedef struct {
 33 |       int nlen;      /* Number of int's in n */
 34 |       UShort *n;     /* Array of unsigned shorts to hold the number */
 35 | } BigNum;
 36 |
 37 | /* Arithmetic function prototypes */
 38 | BigNum * BigNumAdd(const BigNum *a, const BigNum *b);
 39 | BigNum * BigNumSub(const BigNum *a, const BigNum *b);
 40 | BigNum * BigNumMul(const BigNum *a, const BigNum *b);
 41 | BigNum * BigNumDiv(const BigNum *a, const BigNum *b, BigNum **c);
 42 | BigNum * BigNumAlloc(int nlen);
 43 | void     BigNumFree(BigNum *b);
 44 |
 45 | #endif /* BIGNUM__H */
```

TEXT STATISTICS

```
  1414 characters
    45 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    BIGNUM__H       16      17
    BigNum          35      38+     39+     40+     41+     42      43
    BigNumAdd       38
    BigNumAlloc             42
    BigNumDiv       41
    BigNumFree              43
    BigNumMul       40
    BigNumSub       39
    MODULUS         21
    ULong           24
    UShort          23      34
    a       38      39      40      41
    b       38      39      40      41      43
    c       41
    n       34
    nlen            33      42
```

```
/* +++Date last modified: 05-Jul-1997 */

/* BIGNUM1.C
** 
** Routines to do Big Number Arithmetic. These are multi-precision, unsigned
** natural numbers (0, 1, 2, ...). For the storage method, see the BigNum
** typedef in file BIGNUM.H
**
** Released to the public domain by Clifford Rhodes, June 15, 1995, with
** no guarantees of any sort as to accuracy or fitness of use.
*/

#include <stdlib.h>
#include "bignum.h"    /* Typedef for BigNum type */


BigNum * BigNumAdd(const BigNum * a, const BigNum * b)
{
      /* Big Numbers a and b are added. If successful a pointer to the sum is
      ** returned. If unsuccessful, a NULL is returned.
      ** Neither a nor b is changed by the call.
      */

      UShort  carry = 0;
      int     size, la, lb;
      long    tsum;
      BigNum  * sum;

      /* Allocate memory for Big Number sum to be returned... */

      size = max(a->nlen, b->nlen) + 1;
      if ((sum = BigNumAlloc(size)) == NULL)
            return NULL;

      /* Get indexes to last digits in each number (Least significant) */

      la = a->nlen - 1;
      lb = b->nlen - 1;
      size--;

      while (la >= 0 && lb >= 0)    /* While both a and b have data */
      {
            tsum = carry + (long) *(a->n + la) + (long) *(b->n + lb);
            *(sum->n + size) = (UShort) (tsum % (long) MODULUS);
            carry = (tsum / (long) MODULUS) ? 1 : 0;
            la--;
            lb--;
            size--;
      }
      if (lb < 0)                   /* Must see if a still has data */
      {
            while (carry && la >= 0)
            {
                  tsum = carry + (long) *(a->n + la);
                  *(sum->n + size) = (UShort) (tsum % (long) MODULUS);
                  carry = (tsum / (long) MODULUS) ? 1 : 0;
                  la--;
                  size--;
            }
            while (la >= 0)
            {
                  *(sum->n + size) = *(a->n + la);
                  la--;
                  size--;
            }
      }
      else                          /* See if b still has data */
      {
            while (carry && lb >= 0)
            {
                  tsum = carry + (long) *(b->n + lb);
                  *(sum->n + size) = (UShort) (tsum % (long) MODULUS);
                  carry = (tsum / (long) MODULUS) ? 1 : 0;
                  lb--;
                  size--;
            }
            while (lb >= 0)
            {
                  *(sum->n + size) = *(b->n + lb);
                  lb--;
                  size--;
            }
      }
      *(sum->n + size) = carry;

      return sum;
}

BigNum * BigNumSub(const BigNum * a, const BigNum * b)
{
      /* Big Numbers a and b are subtracted. a must be >= to b. A pointer to
      ** the difference (a - b) is returned if successful. If unsuccessful,
      ** a NULL is returned.
      ** Neither a nor b is changed by the call.
      */

      int      size, la, lb, borrow = 0;
      long     tdiff;
      BigNum * diff;

```

```
101 |       /* Allocate memory for Big Number difference to be returned... */
102 |
103 |       if ((diff = BigNumAlloc(a->nlen)) == NULL)
104 |             return NULL;
105 |
106 |       la = a->nlen - 1;
107 |       size = la;
108 |       lb = b->nlen - 1;
109 |
110 |       while (lb >= 0)
111 |       {
112 |             tdiff = (long) *(a->n + la) - (long) *(b->n + lb) - borrow;
113 |             if (tdiff < 0)
114 |             {
115 |                   tdiff += (long) (MODULUS - 1);
116 |                   borrow = 1;
117 |             }
118 |             else  borrow = 0;
119 |             *(diff->n + size) = (UShort) tdiff + borrow;
120 |             la--;
121 |             lb--;
122 |             size--;
123 |       }
124 |       while (la >= 0)          /* Must get rest of a->n */
125 |       {
126 |             tdiff = (long) *(a->n + la) - borrow;
127 |             if (tdiff < 0)
128 |             {
129 |                   tdiff += (long) (MODULUS - 1);
130 |                   borrow = 1;
131 |             }
132 |             else  borrow = 0;
133 |             *(diff->n + size) = (UShort) tdiff + borrow;
134 |             la--;
135 |             size--;
136 |       }
137 |       if (borrow)   /* We've got an underflow here! */
138 |       {
139 |             BigNumFree(diff);
140 |             return NULL;
141 |       }
142 |       return diff;
143 | }
144 |
145 | BigNum * BigNumMul(const BigNum * a, const BigNum * b)
146 | {
147 |       /* Big Numbers a and b are multiplied. A pointer to the product
148 |       ** is returned if successful. If unsuccessful, a NULL is returned.
149 |       ** Neither a nor b is changed by the call.
150 |       */
151 |
152 |       int      size, la, lb, apos, bpos, ppos;
153 |       UShort   carry;
154 |       BigNum * product;
155 |
156 |       /* Allocate memory for Big Number product to be returned... */
157 |
158 |       size = a->nlen + b->nlen;
159 |       if ((product = BigNumAlloc(size)) == NULL)
160 |             return NULL;
161 |
162 |       la = a->nlen - 1;
163 |       lb = b->nlen - 1;
164 |       size--;
165 |
166 |       bpos = lb;
167 |
168 |       while (bpos >= 0)             /* We'll multiply by each digit in b */
169 |       {
170 |             ppos = size + (bpos - lb);    /* Position in product */
171 |
172 |             if (*(b->n + bpos) == 0) /* If zero multiplier, skip this pass */
173 |                   ppos = ppos - la - 1;
174 |             else                    /* Multiply a by next b digit */
175 |             {
176 |                   apos = la;
177 |                   carry = 0;
178 |                   while (apos >= 0) /* Go a digit at a time through a */
179 |                   {
180 |                         ULong temp;
181 |
182 |                         temp = (ULong) *(a->n + apos) *
183 |                               (ULong) *(b->n + bpos) + carry;
184 |
185 |                         /*
186 |                         ** We must add any previous product term in
187 |                         ** this 'column' too
188 |                         */
189 |
190 |                         temp += (ULong) *(product->n + ppos);
191 |
192 |                         /* Now update product term, remembering any carry */
193 |
194 |                         *(product->n + ppos) =
195 |                               (UShort) (temp % (ULong) MODULUS);
196 |                         carry = (UShort) (temp / (ULong) MODULUS);
197 |
198 |                         apos--;
199 |                         ppos--;
200 |                   }
```

```
 201 |                   *(product->n + ppos) = carry;
 202 |               }
 203 |               bpos--;
 204 |       }
 205 |       return product;
 206 | }
```

TEXT STATISTICS

```
  6262 characters
   206 lines
```

LEXICAL STATISTICS

```
    22 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
    37 constants [numeric]
```

SYMBOL TABLE

```
    BigNum          17+     27      89+     99      145+    154
    BigNumAdd       17
    BigNumAlloc             32      103     159
    BigNumFree              139
    BigNumMul       145
    BigNumSub       89
    MODULUS         44      45      55      56      72      73      115     129     195     196
    NULL            32      33      103     104     140     159     160
    ULong           180     182     183     190     195     196
    UShort          24      44      55      72      119     133     153     195     196
    a       17      31      37      43      54      62      89      103     106     112     126     145
            158     162     182
    apos            152     176     178     182     198
    b       17      31      38      43      71      79      89      108     112     145     158     163
            172     183
    borrow          97      112     116     118     119     126     130     132     133     137
    bpos            152     166     168     170     172     183     203
    carry           24      43      45      52      54      56      69      71      73      84      153
            177     183     196     201
    diff            99      103     119     133     139     142
    h       13
    la              25      37      41      43      46      52      54      57      60      62      63
            97      106     107     112     120     124     126     134     152     162     173     176
    lb              25      38      41      43      47      50      69      71      74      77      79
            80      97      108     110     112     121     152     163     166     170
    max             31
    n       43+     44      54      55      62+     71      72      79+     84      112+    119     126
            133     172     182     183     190     194     201
    nlen            31+     37      38      103     106     108     158+    162     163
    ppos            152     170     173+    190     194     199     201
    product         154     159     190     194     201     205
    size            25      31      32      39      44      48      55      58      62      64      72
            75      79      81      84      97      107     119     122     133     135     152     158
            159     164     170
    stdlib          13
    sum             27      32      44      55      62      72      79      84      86
    tdiff           98      112     113     115     119     126     127     129     133
    temp            180     182     190     195     196
    tsum            26      43      44      45      54      55      56      71      72      73
```

```
/* +++Date last modified: 05-Jul-1997 */

/* BIGNUM2.C
**
** Routines to do Big Number Arithmetic. These are multi-precision, unsigned
** natural numbers (0, 1, 2, ...). For the storage method, see the BigNum
** typedef in file BIGNUM.H
**
** Released to the public domain by Clifford Rhodes, June 15, 1995, with
** no guarantees of any sort as to accuracy or fitness of use.
*/

#include <stdlib.h>
#if defined(_MSC_VER)
 #include <memory.h>
#elif defined(__TURBOC__)
 #include <mem.h>
#else
 #include <string.h>
#endif

#include "bignum.h"    /* Typedef for BigNum type */

BigNum * BigNumAlloc(int nlen)
{
      /* Allocates memory for a BigNum object with nlen entries. Returns a
      ** pointer to the memory with the data array initialized to zero if
      ** successful. If not successful, NULL is returned.
      */

      BigNum * big;

      big = (BigNum *) malloc(sizeof(BigNum));
      if (big != NULL)
      {
            big->nlen = nlen;
            if (nlen < 1)
                  big->n = NULL;
            else if ((big->n =
                  (UShort *) calloc(nlen, sizeof(UShort))) == NULL)
            {
                  free(big);
                  return NULL;
            }
      }
      else  return NULL;

      return big;
}

void BigNumFree(BigNum * b)
{
      /* Frees memory used by BigNum object b. */

      if (b != NULL)
      {
            if (b->n != NULL)
                  free(b->n);
            free(b);
      }
}

BigNum * BigNumDiv(const BigNum * a, const BigNum * b, BigNum ** c)
{
      /* Big Number a is divided by Big Number b. If successful a pointer to
      ** the quotient is returned. The user must supply a pointer to a Big
      ** Number pointer, c, to receive the remainder.
      ** If unsuccessful, NULL is returned.
      ** Neither a nor b is changed by the call.
      */

      int      i, j, d, bpos;
      UShort   carry, quo;
      long     m1, m2, m3;
      BigNum * quotient, * atemp, * btemp;

      /* Find non-zero MSB of b, make sure b is not 0 */

      for (bpos = 0; bpos < b->nlen && b->n[bpos] == 0; bpos++)
            ;
      if (bpos == b->nlen)  /* Attempt to divide by zero! */
            return NULL;

      /* Create a copy of b, making sure btemp->n[0] != 0 */

      if ((btemp = BigNumAlloc(b->nlen - bpos)) == NULL)
            return NULL;
      memcpy(btemp->n, b->n + bpos, btemp->nlen * sizeof(UShort));

      /* Create a copy of a, making sure atemp->n[0] == 0 */

      carry = (a->n[0] == 0) ? 0 : 1;
      if ((atemp = BigNumAlloc(a->nlen + carry)) == NULL)
      {
            BigNumFree(btemp);
            return NULL;
      }
      memcpy(atemp->n + carry, a->n, a->nlen * sizeof(UShort));

      /* Allocate memory for quotient and remainder */
```

```
101 |
102 |       if ((quotient = BigNumAlloc(max(1, atemp->nlen - btemp->nlen))) == NULL)
103 |       {
104 |             BigNumFree(atemp);
105 |             BigNumFree(btemp);
106 |             return NULL;
107 |       }
108 |       if ((*c = BigNumAlloc(btemp->nlen)) == NULL)
109 |       {
110 |             BigNumFree(atemp);
111 |             BigNumFree(btemp);
112 |             BigNumFree(quotient);
113 |             return NULL;
114 |       }
115 |       d = MODULUS / (btemp->n[0] + 1);
116 |       for (carry = 0, j = atemp->nlen - 1; j >= 0; j--)
117 |       {
118 |             m1 = ((long) d * (long) *(atemp->n + j)) + (long) carry;
119 |             *(atemp->n + j) = (UShort) (m1 % (long) MODULUS);
120 |             carry = (UShort) (m1 / (long) MODULUS);
121 |       }
122 |       for (carry = 0, j = btemp->nlen - 1; j >= 0; j--)
123 |       {
124 |             m1 = ((long) d * (long) *(btemp->n + j)) + (long) carry;
125 |             *(btemp->n + j) = (UShort) (m1 % (long) MODULUS);
126 |             carry = (UShort) (m1 / (long) MODULUS);
127 |       }
128 |       for (j = 0; j < (atemp->nlen - btemp->nlen); j++)
129 |       {
130 |             if (*btemp->n == *(atemp->n + j))
131 |                   quo = MODULUS - 1;
132 |             else
133 |             {
134 |                   m1 = (long) *(atemp->n + j) * (long) MODULUS;
135 |                   m1 = (m1 + (long) *(atemp->n + j + 1)) / (long) *btemp->n;
136 |                   quo = (UShort) m1;
137 |             }
138 |             m3 = (long) *(atemp->n + j) * (long) MODULUS +
139 |                   (long) *(atemp->n + j + 1);
140 |             do
141 |             {
142 |                   if (btemp->nlen > 1)
143 |                         m1 = (long) quo * (long) *(btemp->n + 1);
144 |                   else  m1 = 0l;
145 |                   m2 = (long) quo * (long) *btemp->n;
146 |                   m2 = (m3 - m2) * (long) MODULUS +
147 |                         (long) *(atemp->n + j + 2);
148 |                   if (m1 > m2)
149 |                         quo--;
150 |             } while (m1 > m2);
151 |
152 |             bpos = btemp->nlen - 1;
153 |             i = j + btemp->nlen;
154 |             m2 = 0l;
155 |             while (i >= j)
156 |             {
157 |                   if (bpos >= 0)
158 |                         m1 = (long) quo * (long) *(btemp->n + bpos);
159 |                   else  m1 = 0l;
160 |                   m3 = (long) *(atemp->n + i) - m1 + m2;
161 |                   if (m3 < 0l)
162 |                   {
163 |                         m2 = m3 / (long) MODULUS;
164 |                         m3 %= (long) MODULUS;
165 |                         if (m3 < 0l)
166 |                         {
167 |                               m3 += (long) MODULUS;
168 |                               m2--;
169 |                         }
170 |                   }
171 |                   else  m2 = 0l;
172 |                   *(atemp->n + i) = (UShort) (m3);
173 |                   bpos--;
174 |                   i--;
175 |             }
176 |             if (m2 == 0l)
177 |                   *(quotient->n + j) = quo;
178 |             else
179 |             {
180 |                   *(quotient->n + j) = quo - 1;
181 |                   bpos = btemp->nlen - 1;
182 |                   i = j + btemp->nlen;
183 |                   carry = 0;
184 |                   while (i >= j)
185 |                   {
186 |                         long tmp;
187 |
188 |                         if (bpos >= 0)
189 |                               carry += *(btemp->n + bpos);
190 |                         tmp = carry + (long) *(atemp->n + i);
191 |                         if (tmp >= (long) MODULUS)
192 |                         {
193 |                               tmp -= (long) MODULUS;
194 |                               carry = 1;
195 |                         }
196 |                         else  carry = 0;
197 |                         *(atemp->n + i) = (UShort) tmp;
198 |                         bpos--;
199 |                         i--;
200 |                   }
```

```
 201 |             }
 202 |       }
 203 |       j = atemp->nlen - btemp->nlen;
 204 |       bpos = 0;
 205 |       carry = 0;
 206 |       while (j < atemp->nlen)
 207 |       {
 208 |             m3 = (long) carry * (long) MODULUS + (long) *(atemp->n + j);
 209 |             *((*c)->n + bpos) = (UShort) (m3 / d);
 210 |             carry = (UShort) (m3 % d);
 211 |             j++;
 212 |             bpos++;
 213 |       }
 214 |       BigNumFree(atemp);   /* Free temporary memory */
 215 |       BigNumFree(btemp);
 216 |
 217 |       return quotient;
 218 | }
```

TEXT STATISTICS

```
  6944 characters
   218 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     1 constants [string]
    40 constants [numeric]
```

SYMBOL TABLE

```
    BigNum          24      31      33+     51      63+     75
    BigNumAlloc             24      86      93      102     108
    BigNumDiv       63
    BigNumFree              51      95      104     105     110     111     112     214     215
    MODULUS         115     119     120     125     126     131     134     138     146     163     164
            167     191     193     208
    NULL            34      38      40      43      46      55      57      82      86      87      93
             96     102     106     108     113
    UShort          40+     73      88      98      119     120     125     126     136     172     197
            209     210
    _MSC_VER        14
    __TURBOC__              16
    a       63      92      93      98+
    atemp           75      93      98      102     104     110     116     118     119     128     130
            134     135     138     139     147     160     172     190     197     203     206     208
            214
    b        51     55      57      58      59      63      79+     81      86      88
    big             31      33      34      36      38      39      42      48
    bpos            72      79+     81      86      88      152     157     158     173     181     188
            189     198     204     209     212
    btemp           75      86      88+     95      102     105     108     111     115     122     124
            125     128     130     135     142     143     145     152     153     158     181     182
            189     203     215
    c        63     108     209
    calloc          40
    carry           73      92      93      98      116     118     120     122     124     126     183
            189     190     194     196     205     208     210
    d        72     115     118     124     209     210
    defined         14      16
    free            42      58      59
    h        13     15      17      19
    i        72     153     155     160     172     174     182     184     190     197     199
    j        72     116+    118     119     122+    124     125     128+    130     134     135     138
            139     147     153     155     177     180     182     184     203     206     208     211
    m1              74      118     119     120     124     125     126     134     135+    136     143
            144     148     150     158     159     160
    m2              74      145     146+    148     150     154     160     163     168     171     176
    m3              74      138     146     160     161     163     164     165     167     172     208
            209     210
    malloc          33
    max             102
    mem             17
    memcpy          88      98
    memory          15
    n        38     39      57      58      79      88+     92      98+     115     118     119     124
            125     130+    134     135+    138     139     143     145     147     158     160     172
            177     180     189     190     197     208     209
    nlen            24      36+     37      40      79      81      86      88      93      98      102+
            108     116     122     128+    142     152     153     181     182     203+    206
    quo             73      131     136     143     145     149     158     177     180
    quotient        75      102     112     177     180     217
    stdlib          13
    string          19
    tmp             186     190     191     193     197
```

```
/* +++Date last modified: 05-Jul-1997 */

/* BIGTEST.C
**
** Routines to test Big Number Arithmetic functions found in BIGNUM.C
**
*/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#if defined(_MSC_VER)
 #include <memory.h>
#elif defined(__TURBOC__)
 #include <mem.h>
#else
 #include <string.h>
#endif

#include "bignum.h"    /* Typedef for BigNum type */

ULong Big2Long(BigNum * b)
{
      /* Converts the big number in b to an unsigned long. This is a support
      ** routine for test purposes only. The length of b should be 3 or less
      ** to avoid overflowing the long.
      */

      int i;
      ULong res = 0L;

      for (i = 0; i < b->nlen; i++)
      {
            res *= (ULong) MODULUS;
            res += (ULong) b->n[i];
      }
      return res;
}

void BigNumRand(BigNum **b)
{
      /* This routine fills the big number pointed to by *b to a random long.
      ** It is a support routine for test purposes only.
      */

      int  i = (*b)->nlen - 1;

      memset((*b)->n, 0, (*b)->nlen * sizeof(UShort));

      if (i < 0)
            return;
      else if (i == 0)
            (*b)->n[0] = rand() % MODULUS;
      else if (i == 1)
      {
            (*b)->n[0] = rand() % MODULUS;
            (*b)->n[1] = rand() % MODULUS;
      }
      else
      {
            (*b)->n[i--] = rand() % MODULUS;
            (*b)->n[i--] = rand() % MODULUS;
            (*b)->n[i] = rand() % 21;
      }
}

int main(void)
{
      /* Test the big number arithmetic routines using random longs. Runs
      ** until an error is encountered or a key pressed.
      ** Gives examples of how to call the routines.
      */

      BigNum *a, *b, *c, *d = NULL, *p, *s;
      ULong al, bl, cl, dl, sl, cnt = 0L;

      /* 'Randomly' seed the rand() generator */

      srand((unsigned int) time(NULL));

      /* Create two big numbers */

      a = BigNumAlloc(3);
      b = BigNumAlloc(3);

      do    /* Loop until a key is pressed... */
      {
            if ((++cnt % 1000L) == 0)     /* Show every 1000 iterations */
                  printf("%ld\n", cnt);
            if (kbhit())            /* If a key pressed--we're outta here! */
            {
                  getch();
                  break;
            }

            /* Make the two big numbers random... */

            BigNumRand(&a);
            BigNumRand(&b);
```

```
101 |
102 |             /* Get their 'long' values... */
103 |
104 |             al = Big2Long(a);
105 |             bl = Big2Long(b);
106 |
107 |             c = BigNumDiv(a, b, &d);        /* Divide a by b */
108 |             if (c == NULL)
109 |             {
110 |                   printf("Error: Divide did not return quotient\n");
111 |                   break;
112 |             }
113 |             if (d == NULL)
114 |             {
115 |                   printf("Error: Divide did not return remainder\n");
116 |                   break;
117 |             }
118 |             cl = Big2Long(c);        /* Get quotient as a long */
119 |             dl = Big2Long(d);        /* Get remainder as a long */
120 |
121 |             if ((al / bl) != cl)
122 |             {
123 |                   printf("Error: Quotient %lu / %lu != %lu\n", al, bl, cl);
124 |                   printf("a = %d%04d%04d   b = %d%04d\n",
125 |                         a->n[0], a->n[1], a->n[2], b->n[0], b->n[1]);
126 |                   break;
127 |             }
128 |             if ((al % bl) != dl)
129 |             {
130 |                   printf("Error: Remainder %lu %c %lu != %lu\n",
131 |                         al, '%', bl, dl);
132 |                   printf("a = %d%04d%04d   b = %d%04d\n",
133 |                         a->n[0], a->n[1], a->n[2], b->n[0], b->n[1]);
134 |                   break;
135 |             }
136 |
137 |             /* Reconstruct BigNum a by multiplying the quotient by b and
138 |             ** adding the remainder.
139 |             */
140 |
141 |             p = BigNumMul(c, b);
142 |             s = BigNumAdd(p, d);
143 |             sl = Big2Long(s);
144 |             if (sl != al)
145 |             {
146 |                   printf("Error: Couldn't reconstruct original "
147 |                         "divisor: %lu != %lu\n", al, sl);
148 |                   break;
149 |             }
150 |             BigNumFree(s);    /* To use s again, must free it first! */
151 |             s = BigNumSub(a, b);
152 |             if (al >= bl)     /* For BigNumSub(), a must be > than b */
153 |             {
154 |                   sl = Big2Long(s);
155 |                   if (sl != (al - bl))
156 |                   {
157 |                         printf("Error: Subtraction error %lu - %lu != %lu\n",
158 |                               al, bl, sl);
159 |                         break;
160 |                   }
161 |             }
162 |             else if (s != NULL)
163 |             {
164 |                   printf("Invalid subtraction %lu - %lu\n", al, bl);
165 |                   break;
166 |             }
167 |
168 |             /* Free all the memory allocated by the arithmetic calls */
169 |
170 |             BigNumFree(c);
171 |             BigNumFree(d);
172 |             BigNumFree(p);
173 |             BigNumFree(s);
174 |
175 |       } while (1);
176 |
177 |       BigNumFree(a);
178 |       BigNumFree(b);
179 |
180 |       return 0;
181 | }
```

TEXT STATISTICS

```
  5081 characters
   181 lines
```

LEXICAL STATISTICS

```
    20 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     1 constants [character]
    12 constants [string]
    28 constants [numeric]
```

SYMBOL TABLE

```
    Big2Long        23     104     105     118     119     143     154
    BigNum          23      41      75
    BigNumAdd      142
    BigNumAlloc             84      85
    BigNumDiv      107
    BigNumFree              150     170     171     172     173     177     178
    BigNumMul      141
    BigNumRand              41      99     100
    BigNumSub      151
    MODULUS         35      54      57      58      62      63
    NULL            75      80     108     113     162
    ULong           23      31      35      36      76
    UShort          49
    _MSC_VER        13
    __TURBOC__              15
    a       75      84      99     104     107     125+    133+    151     177
    al              76     104     121     123     128     131     144     147     152     155     158
           164
    b       23      33      36      41      47      49+     54      57      58      62      63      64
            75      85     100     105     107     125+    133+    141     151     178
    bl              76     105     121     123     128     131     152     155     158     164
    c       75     107     108     118     141     170
    cl              76     118     121     123
    cnt             76      89      90
    conio           11
    d       75     107     113     119     142     171
    defined         13      15
    dl              76     119     128     131
    getch           93
    h        9      10      11      12      14      16      18
    i       30      33+     36      47      51      53      55      62      63      64
    kbhit           91
    main            68
    mem             16
    memory          14
    memset          49
    n       36      49      54      57      58      62      63      64     125+    133+
    nlen            33      47      49
    p       75     141     142     172
    printf          90     110     115     123     124     130     132     146     157     164
    rand            54      57      58      62      63      64
    res             31      35      36      38
    s       75     142     143     150     151     154     162     173
    sl              76     143     144     147     154     155     158
    srand           80
    stdio           10
    stdlib           9
    string          18
    time            12      80
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** void FAR *BigMalloc(Size_T num_elem, Size_T size_elem);
  5 | ** void BigFree(void FAR *ptr);
  6 | */
  7 |
  8 | #ifndef BIG_MALL__H
  9 | #define BIG_MALL__H
 10 |
 11 | #include "extkword.h"
 12 |
 13 | #if (defined(MSDOS) || defined(__MSDOS__)) && !defined(__FLAT__)
 14 |  #if defined(__TURBOC__) || defined(__POWERC) || defined(__ZTC__)
 15 |   #if defined(__ZTC__)
 16 |    #include <dos.h>
 17 |   #else     /* Borland or Mix */
 18 |    #include <alloc.h>
 19 |   #endif
 20 |   typedef unsigned long Size_T;
 21 |   #define BigMalloc(i,n) (void FAR *)farmalloc((Size_T)(i)*(Size_T)(n))
 22 |   #define BigFree(p) farfree(p)
 23 |  #else                                          /* MSC, Watcom    */
 24 |   #include <malloc.h>
 25 |   #include <stddef.h>                          /* for size_t     */
 26 |   typedef size_t Size_T;
 27 |   #define BigMalloc(i,n) (void FAR *)halloc((Size_T)(i),(Size_T)(n))
 28 |   #define BigFree(p) hfree(p)
 29 |  #endif
 30 | #else
 31 |  #include <stdlib.h>
 32 |  #include <stddef.h>                          /* for size_t     */
 33 |  typedef size_t Size_T;
 34 |  #define BigMalloc(i,n) malloc((Size_T)(i)*(Size_T)(n))
 35 |  #define BigFree(p) free(p)
 36 | #endif
 37 |
 38 | #endif /* BIG_MALL__H */
```

TEXT STATISTICS

```
  1145 characters
    38 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    25 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    BIG_MALL__H              8       9
    BigFree         22      28      35
    BigMalloc       21      27      34
    FAR             21      27
    MSDOS           13
    Size_T          20      21+     26      27+     33      34+
    __FLAT__        13
    __MSDOS__       13
    __POWERC        14
    __TURBOC__              14
    __ZTC__         14      15
    alloc           18
    defined         13+     14+     15
    dos             16
    farfree         22
    farmalloc       21
    free            35
    h       16      18      24      25      31      32
    halloc          27
    hfree           28
    i       21+     27+     34+
    malloc          24      34
    n       21+     27+     34+
    p       22+     28+     35+
    size_t          26      33
    stddef          25      32
    stdlib          31
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** BINCOMP -- binary compare
** by Raymond Gardner -- Englewood CO -- 8/92 -- public domain
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "minmax.h"

#define bufsize             8
#define empty_legend        " .."

unsigned char f1buf[bufsize+1], f2buf[bufsize+1];
long prevn;

void putempty(unsigned n)
{
      while (n--)
            printf(empty_legend);
}

void putbl(unsigned n)
{
      while (n--)
            printf(" ");
}

void showbufs(long n, unsigned m, unsigned char *b1, unsigned char *b2)
{
      unsigned i;

      if (n != prevn + bufsize)
            printf("\n");
      prevn = n;
      printf("%08lX ", n);

      if (b1 && b2)
      {
            for (i = 0; i < m; i++)
                  printf(" %02X", b1[i]);
            for (i = m; i < 8; i++)
                  printf("   ");
            putchar(' ');
            for (i = 0; i < m; i++)
            {
                  if (isprint(b1[i]))
                        putchar(b1[i]);
                  else  putchar(' ');
            }
            for (i = m; i < 8; i++)
                  putchar(' ');
            printf(" |");
            for (i = 0; i < m; i++)
            {
                  if (b1[i] != b2[i])
                        printf(" %02X", b2[i]);
                  else  printf(empty_legend);
            }
            for (i = m; i < 8; i++)
                  printf("   ");
            putchar(' ');
            for (i = 0; i < m; i++)
            {
                  if (b1[i] != b2[i] && isprint(b2[i]))
                        putchar(b2[i]);
                  else  putchar(' ');
            }
      }
      else if (b1)
      {
            for (i = 0; i < m; i++)
                  printf(" %02X", b1[i]);
            for (i = m; i < 8; i++)
                  printf("   ");
            putchar(' ');
            for (i = 0; i < m; i++)
            {
                  if (isprint(b1[i]))
                        putchar(b1[i]);
                  else  putchar(' ');
            }
            for (i = m; i < 8; i++)
                  putchar(' ');
            printf(" |");
      }
      else
      {
            putbl(33);
            printf(" |");
            for (i = 0; i < m; i++)
                  printf(" %02X", b2[i]);
            for (i = m; i < 8; i++)
                  printf("   ");
            putchar(' ');
            for (i = 0; i < m; i++)
            {
```

```
101 |                   if (isprint(b2[i]))
102 |                         putchar(b2[i]);
103 |                   else  putchar(' ');
104 |             }
105 |       }
106 |       printf("\n");
107 | }
108 |
109 | long fsize(FILE *fp)
110 | {
111 |       long pos, size;
112 |
113 |       pos = ftell(fp);
114 |       fseek(fp, 0L, SEEK_END);
115 |       size = ftell(fp);
116 |       fseek(fp, pos, SEEK_SET);
117 |       return size;
118 | }
119 |
120 | void bincomp(FILE *f1, FILE *f2)
121 | {
122 |       unsigned m;
123 |       long f1len, f2len, k, n;
124 |
125 |       prevn = -1;
126 |       f1len = fsize(f1);
127 |       f2len = fsize(f2);
128 |       printf("%ld %ld\n", f1len, f2len);
129 |       k = min(f1len, f2len);
130 |       n = 0;
131 |       while (n < k)
132 |       {
133 |             m = (unsigned)min(k - n, (long)bufsize);
134 |             fread(f1buf, 1, m, f1);
135 |             fread(f2buf, 1, m, f2);
136 |             if (memcmp(f1buf, f2buf, m) != 0)
137 |                   showbufs(n, m, f1buf, f2buf);
138 |             n += m;
139 |       }
140 |       while (n < f1len)
141 |       {
142 |             m = (unsigned)min(f1len - n, (long)bufsize);
143 |             fread(f1buf, 1, m, f1);
144 |             showbufs(n, m, f1buf, NULL);
145 |             n += m;
146 |       }
147 |       while (n < f2len)
148 |       {
149 |             m = (unsigned)min(f2len - n, (long)bufsize);
150 |             fread(f2buf, 1, m, f2);
151 |             showbufs(n, m, NULL, f2buf);
152 |             n += m;
153 |       }
154 | }
155 |
156 | int main(int argc, char **argv)
157 | {
158 |       FILE *f1, *f2;
159 |
160 |       if (argc < 3)
161 |       {
162 |             puts("Usage: bincomp f1 f2");
163 |             exit(0);
164 |       }
165 |       printf("%s vs. %s\n", argv[1], argv[2]);
166 |       f1 = fopen(argv[1], "rb");
167 |       f2 = fopen(argv[2], "rb");
168 |       if (f1 && f2)
169 |             bincomp(f1, f2);
170 |       else  puts("can't open file(s)");
171 |       return 0;
172 | }
```

TEXT STATISTICS

```
  4251 characters
   172 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
    10 constants [character]
    23 constants [string]
    33 constants [numeric]
```

SYMBOL TABLE

```
    FILE            109     120+    158
    NULL            144     151
    SEEK_END        114
    SEEK_SET        116
    argc            156     160
    argv            156     165+    166     167
    b1               32      41      44      50      51      59      68      73      76      82      83
    b2               32      41      59      60      68+     69      95     101     102
    bincomp         120     169
    bufsize          14      17+     36     133     142     149
    ctype            11
    empty_legend             15      23      61
    exit            163
    f1              120     126     134     143     158     166     168     169
    f1buf            17     134     136     137     143     144
    f1len           123     126     128     129     140     142
    f2              120     127     135     150     158     167     168     169
    f2buf            17     135     136     137     150     151
    f2len           123     127     128     129     147     149
    fopen           166     167
    fp              109     113     114     115     116
    fread           134     135     143     150
    fseek           114     116
    fsize           109     126     127
    ftell           113     115
    h         8       9      10      11
    i        34      43+     44      45+     48+     50      51      54+     57+     59+     60      63+
             66+     68+     69      75+     76      77+     80+     82      83      86+     94+     95
             96+     99+    101     102
    isprint          50      68      82     101
    k       123     129     131     133
    m        32      43      45      48      54      57      63      66      75      77      80      86
             94      96      99     122     133     134     135     136     137     138     142     143
            144     145     149     150     151     152
    main            156
    memcmp          136
    min             129     133     142     149
    n        20      22      26      28      32      36      38      39     123     130     131     133
            137     138     140     142     144     145     147     149     151     152
    pos             111     113     116
    prevn            18      36      38     125
    printf           23      29      37      39      44      46      56      60      61      64      76
            78      88      93      95      97     106     128     165
    putbl            26      92
    putchar          47      51      52      55      65      69      70      79      83      84      87
            98     102     103
    putempty         20
    puts            162     170
    showbufs         32     137     144     151
    size            111     115     117
    stdio             8
    stdlib            9
    string           10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  BIPORT.C - Port TC/TC++/BC++ code using register pseudovariables
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #ifndef __TURBOC__
 10 |
 11 | #include "biport.h"
 12 |
 13 | union  REGS  BIP_regs_;
 14 | struct SREGS BIP_sregs_;
 15 |
 16 | #ifndef geninterrupt
 17 |
 18 | unsigned PASCAL geninterrupt(int int_no)
 19 | {
 20 |       int86x(int_no, &BIP_regs_, &BIP_regs_, &BIP_sregs_);
 21 |       return BIP_regs_.x.ax;
 22 | }
 23 |
 24 | #endif /* geninterrupt() */
 25 |
 26 | #endif /* __TURBOC__ */
```

TEXT STATISTICS

```
   478 characters
    26 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    BIP_regs_       13      20+     21
    BIP_sregs_              14      20
    PASCAL          18
    REGS            13
    SREGS           14
    __TURBOC__               9
    ax              21
    geninterrupt            16      18
    int86x          20
    int_no          18      20
    x       21
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  BIPORT.H - Port TC/TC++/BC++ code using register pseudovariables
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #ifndef __TURBOC__
 10 |
 11 | #ifndef BIPORT__H
 12 | #define BIPORT__H
 13 |
 14 | #include <dos.h>
 15 | #include "dirport.h"
 16 |
 17 | extern union  REGS  BIP_regs_;
 18 | extern struct SREGS BIP_sregs_;
 19 |
 20 | #define _AX BIP_regs_.x.ax
 21 | #define _BX BIP_regs_.x.bx
 22 | #define _CX BIP_regs_.x.cx
 23 | #define _DX BIP_regs_.x.dx
 24 | #define _AH BIP_regs_.h.ah
 25 | #define _AL BIP_regs_.h.al
 26 | #define _BH BIP_regs_.h.ah
 27 | #define _BL BIP_regs_.h.al
 28 | #define _CH BIP_regs_.h.ah
 29 | #define _CL BIP_regs_.h.al
 30 | #define _DH BIP_regs_.h.ah
 31 | #define _DL BIP_regs_.h.al
 32 | #define _SI BIP_regs_.x.si
 33 | #define _DI BIP_regs_.x.di
 34 | #define _CF BIP_regs_.x.cflag
 35 | #define _FF BIP_regs_.x.flags
 36 | #define _ES BIP_sregs_.es
 37 | #define _CS BIP_sregs_.cs
 38 | #define _SS BIP_sregs_.ss
 39 | #define _DS BIP_sregs_.ds
 40 | #define regload_() segread(&BIP_sregs_)
 41 |
 42 | #ifndef geninterrupt
 43 |  unsigned PASCAL geninterrupt(int);
 44 | #endif
 45 |
 46 | #endif /* BIPORT__H */
 47 |
 48 | #endif /* __TURBOC__ */
```

TEXT STATISTICS

```
  1056 characters
    48 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    30 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    BIPORT__H       11      12
    BIP_regs_       17      20      21      22      23      24      25      26      27      28      29
            30      31      32      33      34      35
    BIP_sregs_              18      36      37      38      39      40
    PASCAL          43
    REGS            17
    SREGS           18
    _AH             24
    _AL             25
    _AX             20
    _BH             26
    _BL             27
    _BX             21
    _CF             34
    _CH             28
    _CL             29
    _CS             37
    _CX             22
    _DH             30
    _DI             33
    _DL             31
    _DS             39
    _DX             23
    _ES             36
    _FF             35
    _SI             32
    _SS             38
    __TURBOC__               9
    ah              24      26      28      30
    al              25      27      29      31
    ax              20
    bx              21
    cflag           34
    cs              37
```

```
cx              22
di              33
dos             14
ds              39
dx              23
es              36
flags           35
geninterrupt            42      43
h       14      24      25      26      27      28      29      30      31
regload_        40
segread         40
si              32
ss              38
x       20      21      22      23      32      33      34      35
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Functions to maintain an arbitrary length array of bits
  5 | */
  6 |
  7 | #include "bitops.h"
  8 |
  9 | char *alloc_bit_array(size_t bits)
 10 | {
 11 |       char *set = calloc((bits + CHAR_BIT - 1) / CHAR_BIT, sizeof(char));
 12 |
 13 |       return set;
 14 | }
 15 |
 16 | int getbit(char *set, int number)
 17 | {
 18 |         set += number / CHAR_BIT;
 19 |         return (*set & (1 << (number % CHAR_BIT))) != 0;    /* 0 or 1   */
 20 | }
 21 |
 22 | void setbit(char *set, int number, int value)
 23 | {
 24 |         set += number / CHAR_BIT;
 25 |         if (value)
 26 |                 *set |= 1 << (number % CHAR_BIT);           /* set bit  */
 27 |         else    *set &= ~(1 << (number % CHAR_BIT));       /* clear bit*/
 28 | }
 29 |
 30 | void flipbit(char *set, int number)
 31 | {
 32 |         set += number / CHAR_BIT;
 33 |         *set ^= 1 << (number % CHAR_BIT);                  /* flip bit */
 34 | }
```

TEXT STATISTICS

```
   848 characters
    34 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    CHAR_BIT        11+     18      19      24      26      27      32      33
    alloc_bit_array          9
    bits             9      11
    calloc          11
    flipbit         30
    getbit          16
    number          16      18      19      22      24      26      27      30      32      33
    set             11      13      16      18      19      22      24      26      27      30      32
          33
    setbit          22
    size_t           9
    value           22      25
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BITCNTS.C - Test program for bit counting functions
**
**  public domain by Bob Stout & Auke Reitsma
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <limits.h>
#include <time.h>
#include <float.h>
#include "bitops.h"

#define ITERS  1500000L
#define FUNCS  8

static int CDECL bit_shifter(long int x);

int main(void)
{
      clock_t start, stop;
      double ct, cmin = DBL_MAX, cmax = 0;
      int i, cminix, cmaxix;
      long j, n;
      static int (* CDECL pBitCntFunc[FUNCS])(long) = {
            bit_count,
            bitcount,
            ntbl_bitcnt,
            ntbl_bitcount,
            btbl_bitcnt,
            BW_btbl_bitcount,
            AR_btbl_bitcount,
            bit_shifter
      };
      static char *text[FUNCS] = {
            "Optimized 1 bit/loop counter",
            "Ratko's mystery algorithm",
            "Recursive bit count by nybbles",
            "Non-recursive bit count by nybbles",
            "Recursive bit count by bytes",
            "Non-recursive bit count by bytes (BW)",
            "Non-recursive bit count by bytes (AR)",
            "Shift and count bits"
      };

      puts("Bit counter algorithm benchmark\n");

      for (i = 0; i < FUNCS; i++)
      {
            start = clock();

            for (j = n = 0; j < ITERS; j++)
                  n += pBitCntFunc[i](j);

            stop = clock();
            ct = (stop - start) / (double)CLOCKS_PER_SEC;
            if (ct < cmin)
            {
                  cmin = ct;
                  cminix = i;
            }
            if (ct > cmax)
            {
                  cmax = ct;
                  cmaxix = i;
            }

            printf("%-38s> Time: %7.3f sec.; Bits: %ld\n", text[i], ct, n);
      }
      printf("\nBest  > %s\n", text[cminix]);
      printf("Worst > %s\n", text[cmaxix]);
      return 0;
}

static int CDECL bit_shifter(long int x)
{
      int i, n;

      for (i = n = 0; x && (i < (sizeof(long) * CHAR_BIT)); ++i, x >>= 1)
            n += (int)(x & 1L);
      return n;
}
```

TEXT STATISTICS

```
  2176 characters
    85 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
    13 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    AR_btbl_bitcount        35
    BW_btbl_bitcount        34
    CDECL           20      28      78
    CHAR_BIT        82
    CLOCKS_PER_SEC          59
    DBL_MAX         25
    FUNCS           18      28      38      51
    ITERS           17      55
    bit_count       29
    bit_shifter             20      36      78
    bitcount        30
    btbl_bitcnt             33
    clock           53      58
    clock_t         24
    cmax            25      65      67
    cmaxix          26      68      74
    cmin            25      60      62
    cminix          26      63      73
    conio           11
    ct              25      59      60      62      65      67      71
    h        9      10      11      12      13      14
    i       26      51+     56      63      68      71      80      82+
    j       27      55+     56
    limits          12
    main            22
    n       27      55      56      71      80      82      83      84
    ntbl_bitcnt             31
    ntbl_bitcount           32
    pBitCntFunc             28      56
    printf          71      73      74
    puts            49
    start           24      53      59
    stdio            9
    stdlib          10
    stop            24      58      59
    text            38      71      73      74
    time            13
    x       20      78      82+     83
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Bit counter by Ratko Tomic
  5 | */
  6 |
  7 | #include "bitops.h"
  8 |
  9 | int CDECL bit_count(long x)
 10 | {
 11 |         int n = 0;
 12 | /*
 13 | ** The loop will execute once for each bit of x set, this is in average
 14 | ** twice as fast as the shift/test method.
 15 | */
 16 |         if (x) do
 17 |               n++;
 18 |         while (0 != (x = x&(x-1))) ;
 19 |         return(n);
 20 | }
 21 |
 22 | #ifdef TEST
 23 |
 24 | #include <stdlib.h>
 25 | #include "snip_str.h"               /* For plural_text() macro    */
 26 |
 27 | main(int argc, char *argv[])
 28 | {
 29 |       long n;
 30 |
 31 |       while(--argc)
 32 |       {
 33 |             int i;
 34 |
 35 |             n = atol(*++argv);
 36 |             i = bit_count(n);
 37 |             printf("%ld contains %d bit%s set\n",
 38 |                   n, i, plural_text(i));
 39 |       }
 40 |       return 0;
 41 | }
 42 |
 43 | #endif /* TEST */
```

TEXT STATISTICS

```
   804 characters
    43 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     3 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    CDECL            9
    TEST            22
    argc            27      31
    argv            27      35
    atol            35
    bit_count        9      36
    h       24
    i       33      36      38+
    main            27
    n       11      17      19      29      35      36      38
    plural_text             38
    printf          37
    stdlib          24
    x        9      16      18+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Bit counter by Ratko Tomic
  5 | */
  6 |
  7 | #include "bitops.h"
  8 |
  9 | int CDECL bitcount(long i)
 10 | {
 11 |       i = ((i & 0xAAAAAAAAL) >>  1) + (i & 0x55555555L);
 12 |       i = ((i & 0xCCCCCCCCL) >>  2) + (i & 0x33333333L);
 13 |       i = ((i & 0xF0F0F0F0L) >>  4) + (i & 0x0F0F0F0FL);
 14 |       i = ((i & 0xFF00FF00L) >>  8) + (i & 0x00FF00FFL);
 15 |       i = ((i & 0xFFFF0000L) >> 16) + (i & 0x0000FFFFL);
 16 |       return (int)i;
 17 | }
 18 |
 19 | #ifdef TEST
 20 |
 21 | #include <stdlib.h>
 22 | #include "snip_str.h"              /* For plural_text() macro    */
 23 |
 24 | main(int argc, char *argv[])
 25 | {
 26 |       long n;
 27 |
 28 |       while(--argc)
 29 |       {
 30 |             int i;
 31 |
 32 |             n = atol(*++argv);
 33 |             i = bitcount(n);
 34 |             printf("%ld contains %d bit%s set\n",
 35 |                   n, i, plural_text(i));
 36 |       }
 37 |       return 0;
 38 | }
 39 |
 40 | #endif /* TEST */
```

TEXT STATISTICS

```
   872 characters
    40 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     3 constants [string]
    16 constants [numeric]
```

SYMBOL TABLE

```
    CDECL            9
    TEST            19
    argc            24      28
    argv            24      32
    atol            32
    bitcount         9      33
    h       21
    i        9      11+     12+     13+     14+     15+     16      30      33      35+
    main            24
    n       26      32      33      35
    plural_text             35
    printf          34
    stdlib          21
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BITCNT_3.C - Bit counting functions using table lookup
**
**  public domain by Auke Reitsma and Bruce Wedding
*/

#include "bitops.h"   /* from Snippets */

/*
**  Bits table
*/

static char bits[256] =
{
      0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4,  /* 0   - 15  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 16  - 31  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 32  - 47  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 48  - 63  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 64  - 79  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 80  - 95  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 96  - 111 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 112 - 127 */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 128 - 143 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 144 - 159 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 160 - 175 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 176 - 191 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 192 - 207 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 208 - 223 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 224 - 239 */
      4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8   /* 240 - 255 */
};

/*
**  Count bits in each nybble
**
**  Note: Only the first 16 table entries are used, the rest could be
**        omitted.
*/

int CDECL ntbl_bitcount(long int x)
{
      return
            bits[ (int) (x & 0x0000000FUL)       ] +
            bits[ (int)((x & 0x000000F0UL) >> 4) ] +
            bits[ (int)((x & 0x00000F00UL) >> 8) ] +
            bits[ (int)((x & 0x0000F000UL) >> 12)] +
            bits[ (int)((x & 0x000F0000UL) >> 16)] +
            bits[ (int)((x & 0x00F00000UL) >> 20)] +
            bits[ (int)((x & 0x0F000000UL) >> 24)] +
            bits[ (int)((x & 0xF0000000UL) >> 28)];
}

/*
**  Count bits in each byte
**
**  by Bruce Wedding, works best on Watcom & Borland
*/

int CDECL BW_btbl_bitcount(long int x)
{
      union
      {
            unsigned char ch[4];
            long y;
      } U;

      U.y = x;

      return bits[ U.ch[0] ] + bits[ U.ch[1] ] +
             bits[ U.ch[3] ] + bits[ U.ch[2] ];
}

/*
**  Count bits in each byte
**
**  by Auke Reitsma, works best on Microsoft, Symantec, and others
*/

int CDECL AR_btbl_bitcount(long int x)
{
      unsigned char * Ptr = (unsigned char *) &x ;
      int Accu ;

      Accu  = bits[ *Ptr++ ];
      Accu += bits[ *Ptr++ ];
      Accu += bits[ *Ptr++ ];
      Accu += bits[ *Ptr ];
      return Accu;
}

#ifdef TEST

#include <stdlib.h>
#include "snip_str.h"              /* For plural_text() macro    */

main(int argc, char *argv[])
{
      long n;
```

```
 101 |
 102 |       while(--argc)
 103 |       {
 104 |             int i;
 105 |
 106 |             n = atol(*++argv);
 107 |             i = BW_btbl_bitcount(n);
 108 |             printf("%ld contains %d bit%s set\n",
 109 |                   n, i, plural_text(i));
 110 |             i = AR_btbl_bitcount(n);
 111 |             printf("%ld contains %d bit%s set\n",
 112 |                   n, i, plural_text(i));
 113 |       }
 114 |       return 0;
 115 | }
 116 |
 117 | #endif /* TEST */
```

TEXT STATISTICS

```
  3344 characters
   117 lines
```

LEXICAL STATISTICS

```
    25 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     4 constants [string]
   278 constants [numeric]
```

SYMBOL TABLE

```
    AR_btbl_bitcount        81     110
    Accu            84      86      87      88      89      90
    BW_btbl_bitcount        61     107
    CDECL           42      61      81
    Ptr             83      86      87      88      89
    TEST            93
    U       67      69      71+     72+
    argc            98     102
    argv            98     106
    atol           106
    bits            15      45      46      47      48      49      50      51      52      71+     72+
            86      87      88      89
    ch              65      71+     72+
    h       95
    i      104     107     109+    110     112+
    main            98
    n      100     106     107     109     110     112
    ntbl_bitcount           42
    plural_text             109     112
    printf          108     111
    stdlib          95
    x       42      45      46      47      48      49      50      51      52      61      69      81
            83
    y       66      69
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BITCNT_4.C - Recursive bit counting functions using table lookup
**
**  public domain by Bob Stout
*/

#include "bitops.h"   /* from Snippets */

static char bits[256] =
{
      0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4,  /* 0   - 15  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 16  - 31  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 32  - 47  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 48  - 63  */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 64  - 79  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 80  - 95  */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 96  - 111 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 112 - 127 */
      1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,  /* 128 - 143 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 144 - 159 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 160 - 175 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 176 - 191 */
      2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,  /* 192 - 207 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 208 - 223 */
      3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,  /* 224 - 239 */
      4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8   /* 240 - 255 */
};

/*
**  Count bits in each nybble
**
**  Note: Only the first 16 table entries are used, the rest could be
**        omitted.
*/

int CDECL ntbl_bitcnt(long x)
{
      int cnt = bits[(int)(x & 0x0000000FL)];

      if (0L != (x >>= 4))
            cnt += ntbl_bitcnt(x);

      return cnt;
}

/*
**  Count bits in each byte
*/

int CDECL btbl_bitcnt(long x)
{
      int cnt = bits[ ((char *)&x)[0] & 0xFF ];

      if (0L != (x >>= 8))
            cnt += btbl_bitcnt(x);
      return cnt;
}

#ifdef TEST

#include <stdlib.h>
#include "snip_str.h"              /* For plural_text() macro    */

main(int argc, char *argv[])
{
      long n;

      while(--argc)
      {
            int i;

            n = atol(*++argv);
            i = btbl_bitcnt(n);
            printf("%ld contains %d bit%s set\n",
                  n, i, plural_text(i));
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2330 characters
    82 lines

LEXICAL STATISTICS

    23 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     3 constants [string]
   265 constants [numeric]

SYMBOL TABLE

    CDECL           38      52
    TEST            61
    argc            66      70
    argv            66      74
    atol            74
    bits            11      40      54
    btbl_bitcnt             52      57      75
    cnt             40      43      45      54      57      58
    h       63
    i       72      75      77+
    main            66
    n       68      74      75      77
    ntbl_bitcnt             38      43
    plural_text             77
    printf          76
    stdlib          63
    x       38      40      42      43      52      54      56      57
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BITFILES.C - reading/writing bit files
**
**  Public domain by Aare Tali
*/

#include <stdlib.h>
#include "bitops.h"

bfile *bfopen(char *name, char *mode)
{
      bfile * bf;

      bf = malloc(sizeof(bfile));
      if (NULL == bf)
            return NULL;
      bf->file = fopen(name, mode);
      if (NULL == bf->file)
      {
            free(bf);
            return NULL;
      }
      bf->rcnt = 0;
      bf->wcnt = 0;
      return bf;
}

int bfread(bfile *bf)
{
      if (0 == bf->rcnt)          /* read new byte */
      {
            bf->rbuf = (char)fgetc(bf->file);
            bf->rcnt = 8;
      }
      bf->rcnt--;
      return (bf->rbuf & (1 << bf->rcnt)) != 0;
}

void bfwrite(int bit, bfile *bf)
{
      if (8 == bf->wcnt)          /* write full byte */
      {
            fputc(bf->wbuf, bf->file);
            bf->wcnt = 0;
      }
      bf->wcnt++;
      bf->wbuf <<= 1;
      bf->wbuf |= bit & 1;
}

void bfclose(bfile *bf)
{
      fclose(bf->file);
      free(bf);
}

#ifdef TEST

void test1(void)
{
      bfile *out;
      bfile *in;
      FILE  *in1;
      FILE  *in2;

      in = bfopen("bitfiles.c", "rb");
      out = bfopen("bitfiles.cc", "wb");
      if ((NULL == in) || (NULL == out))
      {
            printf("Can't open/create test files\n");
            exit(1);
      }
      while (!feof(in->file))
            bfwrite(bfread(in), out);
      bfclose(in);
      bfclose(out);
      in1 = fopen("bitfiles.c", "rb");
      in2 = fopen("bitfiles.cc", "rb");
      if ((NULL == in1) || (NULL == in2))
      {
            printf("Can't open test files for verifying\n");
            exit(1);
      }
      while (!feof(in1) && !feof(in2))
      {
            if (fgetc(in1) != fgetc(in2))
            {
                  printf("Files not identical, copy failed!\n");
                  exit(1);
            }
      }
      if (!feof(in1) || !feof(in2))
      {
            printf("Not same size, copy failed!\n");
            exit(1);
      }
      fclose(in1);
      fclose(in2);
```

```
101 | }
102 |
103 | void test2(void)
104 | {
105 |       FILE  *in1;
106 |       bfile *in2;
107 |       int    ch;
108 |
109 |       in1 = fopen("bitfiles.c", "rb");
110 |       in2 = bfopen("bitfiles.cc", "rb");
111 |       if ((NULL == in1) || (NULL == in2))
112 |       {
113 |             printf("Can't open test files\n");
114 |             exit(1);
115 |       }
116 |       while (!feof(in1) && !feof(in2->file))
117 |       {
118 |             ch = fgetc(in1);
119 |             if (ch < ' ')
120 |                   ch = '.';
121 |             printf(" '%c' ", ch);
122 |             for (ch = 0; ch < 8; ch++)
123 |                   printf("%c", "01"[bfread(in2)]);
124 |             printf("   ");
125 |       }
126 |       fclose(in1);
127 |       bfclose(in2);
128 | }
129 |
130 | main()
131 | {
132 |       test1();
133 |       test2();
134 |       return 0;
135 | }
136 |
137 | #endif /* TEST */
```

TEXT STATISTICS

2895 characters
137 lines

LEXICAL STATISTICS

5 comments [std-C]
0 comments [C++]
4 preprocessor instructions
2 constants [character]
22 constants [string]
18 constants [numeric]

SYMBOL TABLE

```
FILE            65      66     105
NULL            17      18      20      23      70+     81+    111+
TEST            59
bf              14      16      17      19      20      22      25      26      27      30      32
        34+     35      37      38+     41      43      45+     46      48      49      50      53
        55      56
bfclose         53      77      78     127
bfile           12      14      16      30      41      53      63      64     106
bfopen          12      68      69     110
bfread          30      76     123
bfwrite         41      76
bit             41      50
ch             107     118     119     120     121     122+
exit            73      84      91      97     114
fclose          55      99     100     126
feof            75      86+     94+    116+
fgetc           34      88+    118
file            19      20      34      45      55      75     116
fopen           19      79      80     109
fputc           45
free            22      56
h        9
in              64      68      70      75      76      77
in1             65      79      81      86      88      94      99     105     109     111     116
       118     126
in2             66      80      81      86      88      94     100     106     110     111     116
       123     127
main           130
malloc          16
mode            12      19
name            12      19
out             63      69      70      76      78
printf          72      83      90      96     113     121     123     124
rbuf            34      38
rcnt            25      32      35      37      38
stdlib           9
test1           61     132
test2          103     133
wbuf            45      49      50
wcnt            26      43      46      48
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Macros and prototypes for bit operations
**
**  public domain for SNIPPETS by:
**    Scott Dudley
**    Auke Reitsma
**    Ratko Tomic
**    Aare Tali
**    J. Blauth
**    Bruce Wedding
**    Bob Stout
*/

#ifndef BITOPS__H
#define BITOPS__H

#include <stdio.H>
#include <stdlib.h>                             /* For size_t           */
#include <limits.h>                             /* For CHAR_BIT         */
#include "sniptype.h"                           /* For TOBOOL()         */
#include "extkword.h"                           /* For CDECL            */

/*
**  Macros to manipulate bits in any integral data type.
*/

#define BitSet(arg,posn) ((arg) | (1L << (posn)))
#define BitClr(arg,posn) ((arg) & ~(1L << (posn)))
#define BitFlp(arg,posn) ((arg) ^ (1L << (posn)))
#define BitTst(arg,posn) TOBOOL((arg) & (1L << (posn)))

/*
**  Macros to manipulate bits in an array of char.
**  These macros assume CHAR_BIT is one of either 8, 16, or 32.
*/

#define MASK  CHAR_BIT-1
#define SHIFT ((CHAR_BIT==8)?3:(CHAR_BIT==16)?4:8)

#define BitOff(a,x)  ((void)((a)[(x)>>SHIFT] &= ~(1 << ((x)&MASK))))
#define BitOn(a,x)   ((void)((a)[(x)>>SHIFT] |=  (1 << ((x)&MASK))))
#define BitFlip(a,x) ((void)((a)[(x)>>SHIFT] ^=  (1 << ((x)&MASK))))
#define IsBit(a,x)   ((a)[(x)>>SHIFT]        &   (1 << ((x)&MASK)))

/*
**  BITARRAY.C
*/

char *alloc_bit_array(size_t bits);
int   getbit(char *set, int number);
void  setbit(char *set, int number, int value);
void  flipbit(char *set, int number);

/*
**  BITFILES.C
*/

typedef struct  {
      FILE *  file;       /* for stream I/O   */
      char    rbuf;       /* read bit buffer  */
      char    rcnt;       /* read bit count   */
      char    wbuf;       /* write bit buffer */
      char    wcnt;       /* write bit count  */
} bfile;

bfile * bfopen(char *name, char *mode);
int     bfread(bfile *bf);
void    bfwrite(int bit, bfile *bf);
void    bfclose(bfile *bf);

/*
** BITSTRNG.C
*/

void bitstring(char *str, long byze, int biz, int strwid);

/*
**  BSTR_I.C
*/

unsigned int bstr_i(char *cptr);

/*
**  BITCNT_1.C
*/

int CDECL bit_count(long x);

/*
**  BITCNT_2.C
*/

int CDECL bitcount(long i);

/*
**  BITCNT_3.C
*/

```

```
 101 | int CDECL ntbl_bitcount(long int x);
 102 | int CDECL BW_btbl_bitcount(long int x);
 103 | int CDECL AR_btbl_bitcount(long int x);
 104 |
 105 | /*
 106 | **  BITCNT_4.C
 107 | */
 108 |
 109 | int CDECL ntbl_bitcnt(long x);
 110 | int CDECL btbl_bitcnt(long x);
 111 |
 112 | #endif /*  BITOPS__H */
```

TEXT STATISTICS

```
  2569 characters
   112 lines
```

LEXICAL STATISTICS

```
    22 comments [std-C]
     0 comments [C++]
    18 preprocessor instructions
     0 constants [character]
     2 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

```
    AR_btbl_bitcount        103
    BITOPS__H       16       17
    BW_btbl_bitcount        102
    BitClr          30
    BitFlip         44
    BitFlp          31
    BitOff          42
    BitOn           43
    BitSet          29
    BitTst          32
    CDECL           89       95     101     102     103     109     110
    CHAR_BIT        39       40+
    FILE            61
    H       19
    IsBit           45
    MASK            39       42      43      44      45
    SHIFT           40       42      43      44      45
    TOBOOL          32
    a       42+     43+      44+     45+
    alloc_bit_array          51
    arg             29+      30+     31+     32+
    bf              69       70      71
    bfclose         71
    bfile           66       68      69      70      71
    bfopen          68
    bfread          69
    bfwrite         70
    bit             70
    bit_count       89
    bitcount        95
    bits            51
    bitstring       77
    biz             77
    bstr_i          83
    btbl_bitcnt             110
    byze            77
    cptr            83
    file            61
    flipbit         54
    getbit          52
    h       20      21
    i       95
    limits          21
    mode            68
    name            68
    ntbl_bitcnt             109
    ntbl_bitcount           101
    number          52       53      54
    posn            29+      30+     31+     32+
    rbuf            62
    rcnt            63
    set             52       53      54
    setbit          53
    size_t          51
    stdio           19
    stdlib          20
    str             77
    strwid          77
    value           53
    wbuf            64
    wcnt            65
    x       42+     43+      44+     45+     89      101     102     103     109     110
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** bitstring(): print bit pattern of bytes formatted to string.
**
** By J. Blauth, Sept. 1992. Hereby placed into the public domain.
**
** byze:    value to transform to bitstring.
** biz:     count of bits to be shown (counted from lowest bit, can be any
**          even or odd number).
** strwid:  total width the string shall have. Since between every 4 bits a
**          blank (0x20) is inserted (not added after lowest bit), width of
**          bitformat only is (biz+(biz/4-1)). Bits are printed right aligned,
**          positions from highest bit to start of string filled with blanks.
**          If value of strwid smaller than space needed to print all bits,
**          strwid is ignored (e.g.:
**                bitstr(s,b,16,5) results in 19 chars +'\0').
**
**   EXAMPLE:
**   for (j = 1; j <= 16; j++) { bitstring(s, j, j, 16); puts(s); }
**       1:                1
**       2:               10
**       3:              011
**       d: 0 0000 0000 1101
**       e: 00 0000 0000 1110
**       f: 000 0000 0000 1111
*/

#include "bitops.h"

void bitstring(char *str, long byze, int biz, int strwid)
{
      int i, j;

      j = strwid - (biz + (biz >> 2)- (biz % 4 ? 0 : 1));
      for (i = 0; i < j; i++)
            *str++ = ' ';
      while (--biz >= 0)
      {
            *str++ = ((byze >> biz) & 1) + '0';
            if (!(biz % 4) && biz)
                  *str++ = ' ';
      }
      *str = '\0';
}

#ifdef TEST

#include <stdlib.h>

int main(void)
{
      char s[80]; long j;
      for (j = 1L; j <= 16L; j++)
      {
            bitstring(s, (long)j, (int)j, 16);
            printf("%2ld: %s\n", j, s);
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1800 characters
    62 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     4 constants [character]
     2 constants [string]
    12 constants [numeric]

SYMBOL TABLE

    EXIT_SUCCESS            59
    TEST            47
    bitstring       31      56
    biz             31      35+     38      40      41+
    byze            31      40
    h       49
    i       33      36+
    j       33      35      36      53      54+     56+     57
    main            51
    printf          57
    s       53      56      57
    stdlib          49
    str             31      37      40      42      44
    strwid          31      35
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Boyer-Moore-Horspool pattern match
**  Case-insensitive with accented character translation
**
**  public domain by Raymond Gardner 7/92
**
**  limitation: pattern length + subject length must be less than 32767
**
**  10/21/93 rdg  Fixed bug found by Jeff Dunlop
*/
#include <limits.h>                                         /* rdg 10/93 */
#include <stddef.h>
#include <string.h>

typedef unsigned char uchar;

#define LOWER_ACCENTED_CHARS

unsigned char lowervec[UCHAR_MAX+1] = {                     /* rdg 10/93 */
  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
 32,'!','"','#','$','%','&','\'','(',')','*','+',',','-','.','/',
'0','1','2','3','4','5','6','7','8','9',':',';','<','=','>','?',
'@','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z','[','\\',']','^','_',
'`','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z','{','|','}','~',127,
#ifdef LOWER_ACCENTED_CHARS
'c','u','e','a','a','a','a','c','e','e','e','i','i','i','a','a',
'e',145,146,'o','o','o','u','u','y','o','u',155,156,157,158,159,
'a','i','o','u','n','n',166,167,168,169,170,171,172,173,174,175,
#else
128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,
144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,
160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,
#endif
176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,
192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,
208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,
224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,
240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,
};

#define lowerc(c) lowervec[(uchar)(c)]

#define LARGE 32767

static int patlen;
static int skip[UCHAR_MAX+1];                               /* rdg 10/93 */
static int skip2;
static uchar *pat;

void bmha_init(const char *pattern)
{
      int i, j;
      pat = (uchar *)pattern;
      patlen = strlen(pattern);
      for (i = 0; i <= UCHAR_MAX; ++i)                      /* rdg 10/93 */
      {
            skip[i] = patlen;
            for (j = patlen - 1; j >= 0; --j)
            {
                  if (lowerc(i) == lowerc(pat[j]))
                        break;
            }
            if (j >= 0)
                  skip[i] = patlen - j - 1;
            if (j == patlen - 1)
                  skip[i] = LARGE;
      }
      skip2 = patlen;
      for (i = 0; i < patlen - 1; ++i)
      {
            if ( lowerc(pat[i]) == lowerc(pat[patlen - 1]) )
                  skip2 = patlen - i - 1;
      }
}

char *bmha_search(const char *string, const int stringlen)
{
      int i, j;
      char *s;

      i = patlen - 1 - stringlen;
      if (i >= 0)
            return NULL;
      string += stringlen;
      for ( ;; )
      {
            while ((i += skip[((uchar *)string)[i]]) < 0)
                  ;                               /* mighty fast inner loop */
            if (i < (LARGE - stringlen))
                  return NULL;
            i -= LARGE;
            j = patlen - 1;
            s = (char *)string + (i - j);
            while (--j >= 0 && lowerc(s[j]) == lowerc(pat[j]))
                  ;
```

```
 101 |             if ( j < 0 )                                  /* rdg 10/93 */
 102 |                  return s;                                 /* rdg 10/93 */
 103 |             if ( (i += skip2) >= 0 )                      /* rdg 10/93 */
 104 |                  return NULL;                              /* rdg 10/93 */
 105 |        }
 106 | }
```

TEXT STATISTICS

```
  3782 characters
   106 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
   125 constants [character]
     0 constants [string]
   199 constants [numeric]
```

SYMBOL TABLE

```
    LARGE           48      71      94      96
    LOWER_ACCENTED_CHARS            19      30
    NULL            88      95     104
    UCHAR_MAX       21      51      60
    bmha_init       55
    bmha_search             81
    c       46+
    h       13      14      15
    i       57      60+     62      65      69      71      74+     76      77      83      86      87
            92+     94      96      98     103
    j       57      63+     65      68      69      70      83      97      98      99+    101
    limits          13
    lowerc          46      65+     76+     99+
    lowervec        21      46
    pat             53      58      65      76+     99
    patlen          50      59      62      63      69      70      73      74      76      77      86
            97
    pattern         55      58      59
    s       84      98      99     102
    skip            51      62      69      71      92
    skip2           52      73      77     103
    stddef          14
    string          15      81      89      92      98
    stringlen       81      86      89      94
    strlen          59
    uchar           17      46      53      58      92
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Case-Insensitive Boyer-Moore-Horspool pattern match
**
**  Public Domain version by Thad Smith 7/21/1992,
**  based on a 7/92 public domain BMH version by Raymond Gardner.
**
**  This program is written in ANSI C and inherits the compilers
**  ability (or lack thereof) to support non-"C" locales by use of
**  toupper() and tolower() to perform case conversions.
**  Limitation: pattern length + string length must be less than 32767.
**
**  10/21/93 rdg  Fixed bugs found by Jeff Dunlop
*/

#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

void bmhi_init(const char *);
char *bmhi_search(const char *, const int);
void bhmi_cleanup(void);

typedef unsigned char uchar;

#define LARGE 32767             /* flag for last character match    */

static int patlen;              /* # chars in pattern               */
static int skip[UCHAR_MAX+1];   /* skip-ahead count for test chars  */
static int skip2;               /* skip-ahead after non-match with
                                ** matching final character         */
static uchar *pat = NULL;       /* uppercase copy of pattern        */

/*
** bmhi_init() is called prior to bmhi_search() to calculate the
** skip array for the given pattern.
** Error: exit(1) is called if no memory is available.
*/

void bmhi_init(const char *pattern)
{
      int i, lastpatchar;
      patlen = strlen(pattern);

      /* Make uppercase copy of pattern */

      pat = realloc ((void*)pat, patlen);
      if (!pat)
            exit(1);
      else  atexit(bhmi_cleanup);
      for (i=0; i < patlen; i++)
            pat[i] = toupper(pattern[i]);

      /* initialize skip array */

      for ( i = 0; i <= UCHAR_MAX; ++i )                    /* rdg 10/93 */
            skip[i] = patlen;
      for ( i = 0; i < patlen - 1; ++i )
      {
            skip[        pat[i] ] = patlen - i - 1;
            skip[tolower(pat[i])] = patlen - i - 1;
      }
      lastpatchar = pat[patlen - 1];
      skip[        lastpatchar ] = LARGE;
      skip[tolower(lastpatchar)] = LARGE;
      skip2 = patlen;                     /* Horspool's fixed second shift */
      for (i = 0; i < patlen - 1; ++i)
      {
            if ( pat[i] == lastpatchar )
                  skip2 = patlen - i - 1;
      }
}

char *bmhi_search(const char *string, const int stringlen)
{
      int i, j;
      char *s;

      i = patlen - 1 - stringlen;
      if (i >= 0)
            return NULL;
      string += stringlen;
      for ( ;; )
      {
            while ( (i += skip[((uchar *)string)[i]]) < 0 )
                  ;                                 /* mighty fast inner loop */
            if (i < (LARGE - stringlen))
                  return NULL;
            i -= LARGE;
            j = patlen - 1;
            s = (char *)string + (i - j);
            while ( --j >= 0 && toupper(s[j]) == pat[j] )
                  ;
            if ( j < 0 )                                    /* rdg 10/93 */
                  return s;                                 /* rdg 10/93 */
            if ( (i += skip2) >= 0 )                        /* rdg 10/93 */
                  return NULL;                              /* rdg 10/93 */
      }
```

```
 101 | }
 102 |
 103 | void bhmi_cleanup(void)
 104 | {
 105 |       free(pat);
 106 | }
```

TEXT STATISTICS

```
  3379 characters
   106 lines
```

LEXICAL STATISTICS

```
    17 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

LARGE		28	66	67	89	91					
NULL		34	83	90	99						
UCHAR_MAX		31	58								
atexit		52									
bhmi_cleanup			24	52	103						
bmhi_init		22	42								
bmhi_search			23	76							
ctype		20									
exit		51									
free		105									
h	17	18	19	20							
i	44	53+	54+	58+	59	60+	62+	63+	69+	71	72	78
	81	82	87+	89	91	93	98					
j	78	92	93	94+	96							
lastpatchar			44	65	66	67	71					
limits		17										
pat		34	49+	50	54	62	63	65	71	94	105	
patlen		30	45	49	53	59	60	62	63	65	68	69
	72	81	92									
pattern		42	45	54								
realloc		49										
s	79	93	94	97								
skip		31	59	62	63	66	67	87				
skip2		32	68	72	98							
stdlib		18										
string		19	76	84	87	93						
stringlen		76	81	84	89							
strlen		45										
tolower		63	67									
toupper		54	94									
uchar		26	34	87								

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Case-sensitive Boyer-Moore-Horspool pattern match
**
**  public domain by Raymond Gardner 7/92
**
**  limitation: pattern length + string length must be less than 32767
**
**  10/21/93 rdg  Fixed bug found by Jeff Dunlop
*/
#include <limits.h>                                     /* rdg 10/93 */
#include <stddef.h>
#include <string.h>

typedef unsigned char uchar;

#define LARGE 32767

static int patlen;
static int skip[UCHAR_MAX+1];                           /* rdg 10/93 */
static int skip2;
static uchar *pat;

void bmh_init(const char *pattern)
{
          int i, lastpatchar;

          pat = (uchar *)pattern;
          patlen = strlen(pattern);
          for (i = 0; i <= UCHAR_MAX; ++i)              /* rdg 10/93 */
                skip[i] = patlen;
          for (i = 0; i < patlen; ++i)
                skip[pat[i]] = patlen - i - 1;
          lastpatchar = pat[patlen - 1];
          skip[lastpatchar] = LARGE;
          skip2 = patlen;               /* Horspool's fixed second shift */
          for (i = 0; i < patlen - 1; ++i)
          {
                if (pat[i] == lastpatchar)
                      skip2 = patlen - i - 1;
          }
}

char *bmh_search(const char *string, const int stringlen)
{
      int i, j;
      char *s;

      i = patlen - 1 - stringlen;
      if (i >= 0)
            return NULL;
      string += stringlen;
      for ( ;; )
      {
            while ( (i += skip[((uchar *)string)[i]]) < 0 )
                  ;                                 /* mighty fast inner loop */
            if (i < (LARGE - stringlen))
                  return NULL;
            i -= LARGE;
            j = patlen - 1;
            s = (char *)string + (i - j);
            while (--j >= 0 && s[j] == pat[j])
                  ;
            if ( j < 0 )                                /* rdg 10/93 */
                  return s;                             /* rdg 10/93 */
            if ( (i += skip2) >= 0 )                    /* rdg 10/93 */
                  return NULL;                          /* rdg 10/93 */
      }
}
```

TEXT STATISTICS

```
  2196 characters
    70 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
    16 constants [numeric]
```

SYMBOL TABLE

```
    LARGE           18      36      58      60
    NULL            52      59      68
    UCHAR_MAX       21      31
    bmh_init        25
    bmh_search              45
    h       12      13      14
    i       27      31+     32      33+     34+     38+     40      41      47      50      51      56+
            58      60      62      67
    j       47      61      62      63+     65
    lastpatchar             27      35      36      40
    limits          12
    pat             23      29      34      35      40      63
    patlen          20      30      32      33      34      35      37      38      41      50      61
    pattern         25      29      30
    s       48      62      63      66
    skip            21      32      34      36      56
    skip2           22      37      41      67
    stddef          13
    string          14      45      53      56      62
    stringlen       45      50      53      58
    strlen          30
    uchar           16      23      29      56
```

```
/* +++Date last modified: 05-Jul-1997 */

/** BODYMASS.C
 *  Calculate body mass index (BMI) for given height and weight.
 *  According to U.S. federal guidelines, a BMI of 24 or less is
 *  desirable. Anything higher is considered overweight.
 *  Donated to the public domain, October 17, 1996.
 **/

short BodyMassIndex(unsigned short height, unsigned short weight)
{
      /* Returns the Body Mass Index (BMI) for height in inches and
      ** weight in pounds. BMI is weight in kilograms divided by height
      ** in meters squared. Returns -1 if invalid height entered.
      */

      /* Define the metric conversion constants... */

#define LBperKG    2.2046f
#define INCHperM  39.37f
#define CFACTOR   ((INCHperM * INCHperM) / LBperKG)

      /* Make sure height is not 0 and is 'reasonable' (100?) */

      if(height < 1 || height > 100)
            return -1;

      return (short) ((((float) weight * CFACTOR) / (height * height)) +
                      0.5f);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
      short bmi, h, w;

      if(argc < 3)
      {
            printf("Usage: bodymass height-in-inches weight-in-pounds\n");
            return EXIT_FAILURE;
      }
      h = atoi(argv[1]);
      w = atoi(argv[2]);
      if (h < 20 || h > 100)
      {
            printf("Height %d out of range!\n", h);
            return EXIT_FAILURE;
      }
      if (w < 20)
      {
            printf("Weight %d out of range!\n", w);
            return EXIT_FAILURE;
      }
      bmi = BodyMassIndex(h, w);
      printf("The Body Mass Index for height %d inches "
             "and weight %d pounds is %d\n", h, w, bmi);
      if (bmi < 25)
            printf("Congratulations! Your index is within the "
                   "recommended range.\n");
      else
      {
            while((bmi = BodyMassIndex(h, --w)) > 24)
                  ;
            printf("Your index is above the recommended level of 24.\n"
                   "To reach that level your weight must drop to %d "
                   "pounds.\n", w);
      }
      return EXIT_SUCCESS;
}

#endif
```

TEXT STATISTICS

```
  2197 characters
    75 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
    10 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

```
    BodyMassIndex           10      58      66
    CFACTOR         21      28
    EXIT_FAILURE            44      51      56
    EXIT_SUCCESS            72
    INCHperM        20      21+
    LBperKG         19      21
    TEST            32
    argc            37      41
    argv            37      46      47
    atoi            46      47
    bmi             39      58      60      61      66
    h       34      35      39      46      48+     50      58      60      66
    height          10      25+     28+
    main            37
    printf          43      50      55      59      62      68
    stdio           34
    stdlib          35
    w       39      47      53      55      58      60      66      70
    weight          10      28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  BORDCOLR.C - set the border color
**  by: Bob Jarvis
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>

char *usage = "BORDCOLR - sets the border color\n"
              "Parameter: color to set - one of\n"
              "\tBLK - black\n"
              "\tBLU - blue\n"
              "\tGRN - green\n"
              "\tCYN - cyan\n"
              "\tRED - red\n"
              "\tMAG - magenta\n"
              "\tBRN - brown\n"
              "\tLTG - light gray\n"
              "\tDKG - dark gray\n"
              "\tLTB - light blue\n"
              "\tLGN - light green\n"
              "\tLTC - light cyan\n"
              "\tLTR - light red\n"
              "\tLTM - light magenta\n"
              "\tYEL - yellow\n"
              "\tWHT - white";

#define BLACK     0
#define BLUE      1
#define GREEN     2
#define CYAN      3
#define RED       4
#define MAGENTA   5
#define BROWN     6
#define LTGRAY    7
#define DKGRAY    8
#define LTBLUE    9
#define LTGREEN   10
#define LTCYAN    11
#define LTRED     12
#define LTMAGENTA 13
#define YELLOW    14
#define WHITE     15

void set_border_color(int color)
{
      union REGS regs;

      printf("color = %d\n", color);

      regs.h.ah = 0x0B;
      regs.h.bh = 0;
      regs.h.bl = color;

      int86(0x10, &regs, &regs);
}

main(int argc, char *argv[])
{
      int color;

      if(argc < 2)
      {
            printf(usage);
            return EXIT_SUCCESS;
      }

      if(stricmp(argv[1], "BLK") == 0)
            color = BLACK;
      else  if(stricmp(argv[1], "BLU") == 0)
            color = BLUE;
      else  if(stricmp(argv[1], "GRN") == 0)
            color = GREEN;
      else  if(stricmp(argv[1], "CYN") == 0)
            color = CYAN;
      else  if(stricmp(argv[1], "RED") == 0)
            color = RED;
      else  if(stricmp(argv[1], "MAG") == 0)
            color = MAGENTA;
      else  if(stricmp(argv[1], "BRN") == 0)
            color = BROWN;
      else  if(stricmp(argv[1], "LTG") == 0)
            color = LTGRAY;
      else  if(stricmp(argv[1], "DKG") == 0)
            color = DKGRAY;
      else  if(stricmp(argv[1], "LTB") == 0)
            color = LTBLUE;
      else  if(stricmp(argv[1], "LGN") == 0)
            color = LTGREEN;
      else  if(stricmp(argv[1], "LTC") == 0)
            color = LTCYAN;
      else  if(stricmp(argv[1], "LTR") == 0)
            color = LTRED;
      else  if(stricmp(argv[1], "LTM") == 0)
            color = LTMAGENTA;
      else  if(stricmp(argv[1], "YEL") == 0)
```

```
 101 |             color = YELLOW;
 102 |       else  if(stricmp(argv[1], "WHT") == 0)
 103 |             color = WHITE;
 104 |       else
 105 |       {
 106 |             printf(usage);
 107 |             return EXIT_SUCCESS;
 108 |       }
 109 |
 110 |       set_border_color(color);
 111 |       return EXIT_SUCCESS;
 112 | }
```

TEXT STATISTICS

```
  2925 characters
   112 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
     0 constants [character]
    35 constants [string]
    52 constants [numeric]
```

SYMBOL TABLE

```
    BLACK           32      73
    BLUE            33      75
    BROWN           38      85
    CYAN            35      79
    DKGRAY          40      89
    EXIT_SUCCESS            69     107     111
    GREEN           34      77
    LTBLUE          41      91
    LTCYAN          43      95
    LTGRAY          39      87
    LTGREEN         42      93
    LTMAGENTA       45      99
    LTRED           44      97
    MAGENTA         37      83
    RED             36      81
    REGS            51
    WHITE           47     103
    YELLOW          46     101
    ah              55
    argc            62      66
    argv            62      72      74      76      78      80      82      84      86      88      90
            92      94      96      98     100     102
    bh              56
    bl              57
    color           49      53      57      64      73      75      77      79      81      83      85
            87      89      91      93      95      97      99     101     103     110
    dos             11
    h        8       9      10      11      55      56      57
    int86           59
    main            62
    printf          53      68     106
    regs            51      55      56      57      59+
    set_border_color        49     110
    stdio            8
    stdlib           9
    stricmp         72      74      76      78      80      82      84      86      88      90      92
            94      96      98     100     102
    string          10
    usage           13      68     106
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Set or determine the status of the DOS "SET BREAK=" command
  5 | */
  6 |
  7 | #include <dos.h>
  8 | #include "snpdosys.h"
  9 |
 10 | /*
 11 | **  Returns status of DOS "SET BREAK" command
 12 | */
 13 |
 14 | int isBreakOn(void)
 15 | {
 16 |       union REGS regs;
 17 |
 18 |       regs.x.ax = 0x3300;
 19 |       intdos(&regs, &regs);
 20 |       return (int)regs.h.dl;
 21 | }
 22 |
 23 | void setBreak(int OnOff)      /* Off = 0, On = 1      */
 24 | {
 25 |       union REGS regs;
 26 |
 27 |       regs.x.ax = 0x3301;
 28 |       regs.h.dl = OnOff;
 29 |       intdos(&regs, &regs);
 30 | }
```

TEXT STATISTICS

```
   532 characters
    30 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    OnOff           23      28
    REGS            16      25
    ax              18      27
    dl              20      28
    dos              7
    h        7      20      28
    intdos          19      29
    isBreakOn       14
    regs            16      18      19+     20      25      27      28      29+
    setBreak        23
    x       18      27
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Public Domain mode 13h Bresenham line/circle algorithms
** By Brian Dessent
**
** Circle algorithm and other stuff rewritten by Kurt Kuzba
**
** Written for Borland, modified for others by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include "bresnham.h"
#include "mk_fp.h"

/* uses BIOS to set video mode */

void setmode(int mode)
{
      union REGS r;

      r.x.ax = mode;
      int86(0x10, &r, &r);
}


/* plots a dot at (x, y) with color c */

void plotdot(int x, int y, char c)
{
      register char far *addr;

      if(x < 0 || x > MAX_X || y < 0 || y > MAX_Y)
            return;

      addr = MK_FP(0xa000, (SCREEN_WIDTH * y) + x);
      *addr = c;
}


/* draws a line from (x, y) to (x2, y2) in color c */

void bresenham_line(int x, int y, int x2, int y2, char c)
{
      int i, steep = 0, sx, sy, dx, dy, e;

      dx = abs(x2 - x);
      sx = ((x2 - x) > 0) ? 1 : -1;
      dy = abs(y2 - y);
      sy = ((y2 - y) > 0) ? 1 : -1;

      if(dy > dx)
      {
            steep =  x;   x =  y;   y = steep;  /* swap  x and  y */
            steep = dx;  dx = dy;  dy = steep;  /* swap dx and dy */
            steep = sx;  sx = sy;  sy = steep;  /* swap sx and sy */
            steep = 1;
      }

      e = 2 * dy - dx;
      for(i = 0; i < dx; i++)
      {
            if(steep)
                  plotdot(y, x, c);
            else  plotdot(x, y, c);
            while(e >= 0)
            {
                  y += sy;
                  e -= 2 * dx;
            }
            x += sx;
            e += 2 * dy;
      }
      plotdot(x2, y2, c);
}

/* draws a circle at (xc, yc) with radius r in color c
**
** note: the scaling factor of (SCREEN_WIDTH / SCREEN_HEIGHT) is used when
** updating d.  This makes round circles.  If you want ellipses, you can
** modify that ratio.
*/

void bresenham_circle(int xc, int yc, int r, char c)
{
      int x = 0, d = 2 * (1 - r), w = 2 * SCREEN_WIDTH / SCREEN_HEIGHT;

      while(r >= 0)
      {
            plotdot(xc + x, yc + r, c);
            plotdot(xc + x, yc - r, c);
            plotdot(xc - x, yc + r, c);
            plotdot(xc - x, yc - r, c);
            if (d + r > 0)
                  d -= (w * --r) - 1;
            if (x > d)
                  d += (2 * ++x) + 1;
```

```
101 |       }
102 | }
103 |
104 | #ifdef TEST
105 |
106 | #ifndef __TURBOC__
107 |  #include "bc_rand.h"
108 | #else
109 |  #include <time.h>
110 | #endif
111 |
112 | /* draws random lines and circles until a key is pressed in mode 13h */
113 | /* (draws in colors 0 - 63 only) */
114 |
115 | int main()
116 | {
117 |       int i=0;
118 |
119 |       randomize();
120 |       setmode(0x13);
121 |       while(!kbhit())
122 |       {
123 |             bresenham_line(random(SCREEN_WIDTH), random(SCREEN_HEIGHT),
124 |                   random(SCREEN_WIDTH), random(SCREEN_HEIGHT), i = ++i % 64);
125 |             bresenham_circle(random(SCREEN_WIDTH), random(SCREEN_HEIGHT),
126 |                   random(50), i = ++i % 64);
127 |       }
128 |       getch();
129 |       setmode(0x03);  /* set to color text mode, clearing screen */
130 |       return 0;
131 | }
132 |
133 | #endif /* TEST */
```

TEXT STATISTICS

```
  3162 characters
   133 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     3 constants [string]
    33 constants [numeric]
```

SYMBOL TABLE

```
    MAX_X           36
    MAX_Y           36
    MK_FP           39
    REGS            23
    SCREEN_HEIGHT           89     123     124     125
    SCREEN_WIDTH            39      89     123     124     125
    TEST           104
    __TURBOC__             106
    abs             50      52
    addr            34      39      40
    ax              25
    bresenham_circle        87     125
    bresenham_line          46     123
    c       32      40      46      67      68      77      87      93      94      95      96
    conio           15
    d       89      97      98      99     100
    dos             14
    dx              48      50      55      58+     63      64      72
    dy              48      52      55      58+     63      75
    e       48      63      69      72      75
    far             34
    getch          128
    h       12      13      14      15     109
    i       48      64+    117     124+    126+
    int86           26
    kbhit          121
    main           115
    mode            21      25
    plotdot         32      67      68      77      93      94      95      96
    r       23      25      26+     87      89      91      93      94      95      96      97      98
    random         123+    124+    125+    126
    randomize      119
    setmode         21     120     129
    stdio           12
    stdlib          13
    steep           48      57+     58+     59+     60      66
    sx              48      51      59+     74
    sy              48      53      59+     71
    time           109
    w       89      98
    x       25      32      36+     39      46      50      51      57+     67      68      74      89
            93      94      95      96      99     100
    x2              46      50      51      77
    xc              87      93      94      95      96
    y       32      36+     39      46      52      53      57+     67      68      71
    y2              46      52      53      77
    yc              87      93      94      95      96
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  Header file for Brian Dessent's Bresenham line/circle algorithms
   5 | */
   6 |
   7 | #ifndef BRESNHAM__H
   8 | #define BRESNHAM__H
   9 |
  10 | #define SCREEN_WIDTH 320
  11 | #define SCREEN_HEIGHT 200
  12 | #define MAX_X (SCREEN_WIDTH-1)
  13 | #define MAX_Y (SCREEN_HEIGHT-1)
  14 |
  15 | /* prototypes */
  16 |
  17 | void setmode(int mode);
  18 | void plotdot(int x, int y, char c);
  19 | void bresenham_line(int x, int y, int x2, int y2, char c);
  20 | void bresenham_circle(int xc, int yc, int r, char c);
  21 |
  22 | #endif /* BRESNHAM__H */
```

TEXT STATISTICS

```
   513 characters
    22 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     0 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    BRESNHAM__H              7       8
    MAX_X           12
    MAX_Y           13
    SCREEN_HEIGHT           11      13
    SCREEN_WIDTH            10      12
    bresenham_circle        20
    bresenham_line          19
    c       18      19      20
    mode            17
    plotdot         18
    r       20
    setmode         17
    x       18      19
    x2              19
    xc              20
    y       18      19
    y2              19
    yc              20
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Make an ascii binary string into an integer.
  5 | **
  6 | **  Public domain by Bob Stout
  7 | */
  8 |
  9 | #include <string.h>
 10 | #include "bitops.h"
 11 |
 12 | unsigned int bstr_i(char *cptr)
 13 | {
 14 |       unsigned int i, j = 0;
 15 |
 16 |       while (cptr && *cptr && strchr("01", *cptr))
 17 |       {
 18 |             i = *cptr++ - '0';
 19 |             j <<= 1;
 20 |             j |= (i & 0x01);
 21 |       }
 22 |       return(j);
 23 | }
 24 |
 25 | #ifdef TEST
 26 |
 27 | #include <stdlib.h>
 28 |
 29 | int main(int argc, char *argv[])
 30 | {
 31 |       char *arg;
 32 |       unsigned int x;
 33 |
 34 |       while (--argc)
 35 |       {
 36 |             x = bstr_i(arg = *++argv);
 37 |             printf("Binary %s = %d = %04Xh\n", arg, x, x);
 38 |       }
 39 |       return EXIT_SUCCESS;
 40 | }
 41 |
 42 | #endif /* TEST */
```

TEXT STATISTICS

```
   739 characters
    42 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
     3 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS            39
    TEST            25
    arg             31      36      37
    argc            29      34
    argv            29      36
    bstr_i          12      36
    cptr            12      16+     18
    h        9      27
    i       14      18      20
    j       14      19      20      22
    main            29
    printf          37
    stdlib          27
    strchr          16
    string           9
    x       32      36      37+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  CAL - a calendar for DOS
**
**  a public domain demo using Ray Gardner's SCALDATE.C scalar date functions
**  by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include "scaldate.h"

/*
**  calendar generation information
*/

int days[12]    = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
char *month[12] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                   "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
char *daynames[8] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

/*
**  box drawing stuff
*/

#if defined(MSDOS) || defined(_WIN32)
 const char *topborder = "\xd5\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xb8";
 const char *midborder = "\xc6\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xb5";
 const char *botborder = "\xd4\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd\xcd"
                         "\xcd\xcd\xcd\xcd\xbe";
 const char *line = "\xb3";
#else
 const char *line = "";
#endif

/*
**  tell 'em they messed up
*/

void usage(void)
{
      puts("Usage: CAL [m y]");
      puts("where: m = month (1 - 12)");
      puts("       y = year (1 - 99, 1800 - 3000)");
      exit(-1);
}

/*
**  here's where the real work's done
*/

int main(int argc, char *argv[])
{
      int day, day_1, numdays, i, j;
      unsigned yr, mo;

      if (argc < 3 && argc > 1)
            usage();

      if (argc >= 3)
      {
            yr = atoi(argv[2]);
            mo = atoi(argv[1]);
      }
      else
      {
            long dnow = today();
            unsigned dy;

            scalar_to_ymd(dnow, &yr, &mo, &dy);
      }

      if (!mo || 12 < mo)
            usage();

      if (100 > yr)
            yr += 1900;

      if (3000 < yr || 1800 > yr)
            usage();

      for (i = 0, mo -= 1; i < 3; ++i, ++mo)
      {
            if (!mo)
            {
                  mo = 12;
                  --yr;
            }
            if (12 < mo)
            {
                  mo = 1;
                  ++yr;
            }
            numdays = days[mo - 1];
```

```
101 |             if (2 == mo && isleap(yr))
102 |                   ++numdays;
103 |             day_1 = (int)((ymd_to_scalar(yr, mo, 1) - (long)ISO_CAL) % 7L);
104 |
105 | #if defined(MSDOS) || defined(_WIN32)
106 |             if (!i)
107 |                   puts(topborder);
108 | #endif
109 |             fputs(line, stdout);
110 |             for (j = 0; j < 7; )
111 |             {
112 |                   fputs(daynames[ISO_CAL + j], stdout);
113 |                   if (7 != ++j)
114 |                         fputc(' ', stdout);
115 |             }
116 |             printf("%s < %s, %d\n%s", line, month[mo - 1], yr, line);
117 |
118 |             for (day = 0; day < day_1; ++day)
119 |                   fputs("    ", stdout);
120 |             for (day = 1; day <= numdays; ++day, ++day_1, day_1 %= 7)
121 |             {
122 |                   if (!day_1 && 1 != day)
123 |                         printf("\b%s\n%s", line, line);
124 |                   printf("%3d ", day);
125 |             }
126 |             for ( ; day_1; ++day_1, day_1 %= 7)
127 |                   fputs("    ", stdout);
128 | #if defined(MSDOS) || defined(_WIN32)
129 |             printf("\b%s\n", line);
130 |             if (2 > i)
131 |                   puts(midborder);
132 |             else  puts(botborder);
133 | #else
134 |             fputc('\n', stdout);
135 | #endif
136 |       }
137 |       return 0;
138 | }
```

TEXT STATISTICS

```
  3688 characters
   138 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     2 constants [character]
    41 constants [string]
    47 constants [numeric]
```

SYMBOL TABLE

```
    ISO_CAL        103     112
    MSDOS           27     105     128
    _WIN32          27     105     128
    argc            58      63+     66
    argv            58      68      69
    atoi            68      69
    botborder       34     132
    day             60     118+    120+    122     124
    day_1           60     103     118     120+    122     126+
    daynames        21     112
    days            18     100
    defined         27+    105+    128+
    dnow            73      76
    dy              74      76
    exit            51
    fputc          114     134
    fputs          109     112     119     127
    h       10      11
    i       60      88+    106     130
    isleap         101
    j       60     110+    112     113
    line            37      39     109     116+    123+    129
    main            58
    midborder       31     131
    mo              61      69      76      79+     88+     90      92      95      97     100     101
          103      116
    month           19     116
    numdays         60     100     102     120
    printf         116     123     124     129
    puts            48      49      50     107     131     132
    scalar_to_ymd           76
    stdio           10
    stdlib          11
    stdout         109     112     114     119     127     134
    today           73
    topborder       28     107
    usage           46      64      80      86
    ymd_to_scalar          103
    yr              61      68      76      82      83      85+     93      98     101     103     116
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  public domain demo by Bob Stout
  5 | */
  6 |
  7 | #ifndef CAST__H
  8 | #define CAST__H
  9 |
 10 | #define CAST(new_type,old_object) (*((new_type *)(&(old_object))))
 11 |
 12 | #if 0
 13 | ************************************************************************
 14 | *                                                                      *
 15 | *       /* Example of CAST macro at work        */                    *
 16 | *                                                                      *
 17 | *       union {                                                        *
 18 | *               char    ch[4];                                         *
 19 | *               int     i[2];                                          *
 20 | *       } my_union;                                                    *
 21 | *                                                                      *
 22 | *       long    longvar;                                               *
 23 | *                                                                      *
 24 | *               longvar = (long)my_union;       /* Illegal cast */     *
 25 | *               longvar = CAST(long, my_union); /* Legal cast   */     *
 26 | *                                                                      *
 27 | ************************************************************************
 28 | #endif  /* 0 */
 29 |
 30 | #endif /* CAST__H */
```

TEXT STATISTICS

```
  1370 characters
    30 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     0 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    CAST            10      25
    CAST__H          7       8
    ch              18
    i       19
    longvar         22      24      25
    my_union        20      24      25
    new_type        10+
    old_object              10+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *    ccard - credit card number validation
 *    1994 Peter Miller
 *    Public Domain
 *
 *    This program is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * MANIFEST: functions to validate credit card numbers
 *
 * derived from code by
 *    Chris Stone <cstone@hms.com>
 *    The High Mountain Software Internet Gateway
 *
 * translated to C by
 *    Peter Miller, 28-Oct-94
 *    This source is hereby placed in the Public Domain.
 *    Please leave my name on it,
 *    and document changes in this header block.
 *
 * NO WARRANTY
 *
 *    BECAUSE THE PROGRAM IS IN THE PUBLIC DOMAIN, THERE IS NO
 *    WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE
 *    LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE AUTHORS
 *    AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT
 *    WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING,
 *    BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 *    AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO
 *    THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD
 *    THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL
 *    NECESSARY SERVICING, REPAIR OR CORRECTION.
 *
 *    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
 *    WRITING WILL ANY AUTHOR, OR ANY OTHER PARTY WHO MAY MODIFY
 *    AND/OR REDISTRIBUTE THE PROGRAM, BE LIABLE TO YOU FOR DAMAGES,
 *    INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 *    DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM
 *    (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
 *    RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES
 *    OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
 *    PROGRAMS), EVEN IF SUCH AUTHOR OR OTHER PARTY HAS BEEN ADVISED
 *    OF THE POSSIBILITY OF SUCH DAMAGES.
 */

#include <ctype.h>
#include <string.h>

#include "ccard.h"

#define SIZEOF(a) (sizeof(a) / sizeof((a)[0]))
#define ENDOF(a) ((a) + SIZEOF(a))

#define MINLEN 12
#define MAXLEN (MINLEN + 16)
#define L(n) (1 << ((n) - MINLEN))

static int all_numeric (char *, char *, int);


char *ccard_type_name(ccard_type_ty n)
{
      static char *name[] =
      {
            "unknown",
            "Mastercard",
            "Visa",
            "American Express",
            "Diners Club/Carte Blanche",
            "Discover",
            "enRoute",
            "Japanese Credit Bureau",
            "Australian Bankcard",
      };

      if (n < 0 || n >= SIZEOF(name))
            n = 0;
      return name[n];
}


char *ccard_error_name(ccard_error_ty n)
{
      static char *name[] =
      {
            "no error",
            "card type unknown",
            "card number contains non numeric characters",
            "card number is far too long",
            "card number is the wrong length",
            "checksum incorrect",
      };

      if (n < 0 || n >= SIZEOF(name))
            return "unknown";
      return name[n];
}
```

```
101 |
102 |
103 | static int verify_checksum(char *credit_card)
104 | {
105 |       char        *cp;
106 |       int         dbl;
107 |       int         check_sum;
108 |
109 |       /*
110 |        * This checksum algorithm has a name,
111 |        * but I can't think of it.
112 |        */
113 |       check_sum = 0;
114 |       dbl = 0;
115 |       /* assert(credit_card[0]); */
116 |       cp = credit_card + strlen(credit_card) - 1;
117 |       while (cp >= credit_card)
118 |       {
119 |             int         c;
120 |
121 |             c = *cp-- - '0';
122 |             if (dbl)
123 |             {
124 |                   c *= 2;
125 |                   if (c >= 10)
126 |                         c -= 9;
127 |             }
128 |             check_sum += c;
129 |             dbl = !dbl;
130 |       }
131 |
132 |       return ((check_sum % 10) == 0);
133 | }
134 |
135 |
136 |
137 | static int all_numeric(char *s1, char *s2, int max)
138 | {
139 |       while (*s1)
140 |       {
141 |             if (isspace(*s1) || *s1 == '-')
142 |             {
143 |                   ++s1;
144 |                   continue;
145 |             }
146 |             if (!isdigit(*s1))
147 |                   return ccard_error_non_numeric;
148 |             if (max <= 0)
149 |                   return ccard_error_too_long;
150 |             *s2++ = *s1++;
151 |             --max;
152 |       }
153 |       *s2 = 0;
154 |       return ccard_error_none;
155 | }
156 |
157 |
158 | ccard_error_ty ccard_valid(char *credit_card_in, ccard_type_ty *card_type)
159 | {
160 |       typedef struct table_ty table_ty;
161 |       struct table_ty
162 |       {
163 |             char        *prefix;
164 |             int         length_mask;
165 |             ccard_type_ty     type;
166 |             int         checksum;
167 |       };
168 |
169 |       static table_ty table[] =
170 |       {
171 |             { "1800", L(15), ccard_type_jcb, 1, },
172 |             { "2014", L(15), ccard_type_enroute, 0, },
173 |             { "2131", L(15), ccard_type_jcb, 1, },
174 |             { "2149", L(15), ccard_type_enroute, 0, },
175 |             { "300",  L(14), ccard_type_diners, 1, },
176 |             { "301",  L(14), ccard_type_diners, 1, },
177 |             { "302",  L(14), ccard_type_diners, 1, },
178 |             { "303",  L(14), ccard_type_diners, 1, },
179 |             { "304",  L(14), ccard_type_diners, 1, },
180 |             { "305",  L(14), ccard_type_diners, 1, },
181 |             { "34",   L(15), ccard_type_amex, 1, },
182 |             { "36",   L(14), ccard_type_diners, 1, },
183 |             { "37",   L(15), ccard_type_amex, 1, },
184 |             { "38",   L(14), ccard_type_diners, 1, },
185 |             { "3",    L(16), ccard_type_jcb, 1, },
186 |             { "4",    L(13)|L(16), ccard_type_visa, 1, },
187 |             { "51",   L(16), ccard_type_mastercard, 1, },
188 |             { "52",   L(16), ccard_type_mastercard, 1, },
189 |             { "53",   L(16), ccard_type_mastercard, 1, },
190 |             { "54",   L(16), ccard_type_mastercard, 1, },
191 |             { "55",   L(16), ccard_type_mastercard, 1, },
192 |             { "56",   L(16), ccard_type_bankcard, 1, },
193 |             { "6011", L(16), ccard_type_discover, 1, },
194 |       };
195 |       table_ty    *tp;
196 |       char        credit_card[MAXLEN + 1];
197 |       ccard_error_ty    err;
198 |       int         len;
199 |
200 |       /*
```

```
201 |        * copy the number, eliding spaces
202 |        * defer any errors until after we have tried to guess the card type
203 |        */
204 |       err = all_numeric(credit_card_in, credit_card, MAXLEN);
205 |
206 |       /*
207 |        * look for the card prefix in the table
208 |        * to determine the card type
209 |        */
210 |       for (tp = table; tp < ENDOF(table); ++tp)
211 |       {
212 |             if (!memcmp(tp->prefix, credit_card, strlen(tp->prefix)))
213 |                   break;
214 |       }
215 |       if (tp >= ENDOF(table))
216 |       {
217 |             *card_type = ccard_type_unknown;
218 |             return ccard_error_type_unknown;
219 |       }
220 |       *card_type = tp->type;
221 |       if (err != ccard_error_none)
222 |             return err;
223 |
224 |       /*
225 |        * set the card type, then check the length
226 |        */
227 |       /* assert(tp->correct_length <= MAXLEN); */
228 |       len = strlen(credit_card);
229 |       if (len < MINLEN || (L(len) & tp->length_mask) == 0)
230 |             return ccard_error_length_incorrect;
231 |
232 |       /*
233 |        * checksum
234 |        */
235 |       if (tp->checksum && !verify_checksum(credit_card))
236 |             return ccard_error_checksum;
237 |
238 |       /*
239 |        * no errors found
240 |        */
241 |       return ccard_error_none;
242 | }
243 |
244 | #ifdef TEST
245 | /*
246 |  *    ccard - credit card number validation
247 |  *    1994 Peter Miller
248 |  *    Public Domain
249 |  *
250 |  *    This program is distributed in the hope that it will be useful,
251 |  *    but WITHOUT ANY WARRANTY; without even the implied warranty of
252 |  *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
253 |  *
254 |  * MANIFEST: program entry point
255 |  */
256 |
257 | #include <stdio.h>
258 | #include <stdlib.h>
259 | #include <string.h>
260 |
261 | #include "ccardplv.h"
262 |
263 | static char *progname;
264 |
265 |
266 | static int suffix(char *s1, char *s2)
267 | {
268 |       int len1 = strlen(s1);
269 |       int len2 = strlen(s2);
270 |       return (len2 < len1 && !memcmp(s1 + len1 - len2, s2, len2));
271 | }
272 |
273 |
274 | static void print_version(char *s)
275 | {
276 |       char        *ep;
277 |
278 |       for (;;)
279 |       {
280 |             ep = strrchr(s, '/');
281 |             if (!ep)
282 |                   break;
283 |             if (ep > s && !ep[1])
284 |             {
285 |                   *ep = 0;
286 |                   continue;
287 |             }
288 |             s = ep + 1;
289 |             break;
290 |       }
291 |       progname = s;
292 |
293 |       fprintf(stderr, "%s version %s\n", progname, PATCHLEVEL);
294 | }
295 |
296 |
297 | int main(int argc, char *argv[])
298 | {
299 |       int         j;
300 |
```

```
301 |       print_version(argv[0]);
302 |       for (j = 1; j < argc; ++j)
303 |       {
304 |             ccard_type_ty     type;
305 |             ccard_error_ty    err;
306 |             char        *s;
307 |
308 |             err = ccard_valid(argv[j], &type);
309 |             if (err)
310 |             {
311 |                   if (type != ccard_type_unknown)
312 |                   {
313 |                         fprintf
314 |                         (
315 |                               stderr,
316 |                               "%s: %s: %s (%s)\n",
317 |                               progname,
318 |                               argv[j],
319 |                               ccard_error_name(err),
320 |                               ccard_type_name(type)
321 |                         );
322 |                   }
323 |                   else
324 |                   {
325 |                         fprintf
326 |                         (
327 |                               stderr,
328 |                               "%s: %s: %s\n",
329 |                               progname,
330 |                               argv[j],
331 |                               ccard_error_name(err)
332 |                         );
333 |                   }
334 |                   exit(1);
335 |             }
336 |             printf("\"%s\" is a", argv[j]);
337 |             s = ccard_type_name(type);
338 |             if (strchr("AEIOUaeiou", s[0]))
339 |                   printf("n");
340 |             printf(" %s", s);
341 |             if (!suffix(s, "card"))
342 |                   printf(" card");
343 |             printf("\n");
344 |       }
345 |       exit(0);
346 |       return 0;
347 | }
348 |
349 | #endif /* TEST */
```

TEXT STATISTICS

10075 characters
349 lines

LEXICAL STATISTICS

12 comments [std-C]
0 comments [C++]
14 preprocessor instructions
3 constants [character]
51 constants [string]
75 constants [numeric]

SYMBOL TABLE

```
ENDOF            55     210     215
L       59     171     172     173     174     175     176     177     178     179     180     181
       182     183     184     185     186+    187     188     189     190     191     192     193
       229
MAXLEN           58     196     204
MINLEN           57      58      59     229
PATCHLEVEL              293
SIZEOF           54      55      79      97
TEST            244
a       54+     55+
all_numeric              61     137     204
argc            297     302
argv            297     301     308     318     330     336
c      119     121     124     125     126     128
card_type       158     217     220
ccard_error_checksum             236
ccard_error_length_incorrect             230
ccard_error_name         85     319     331
ccard_error_non_numeric          147
ccard_error_none        154     221     241
ccard_error_too_long             149
ccard_error_ty           85     158     197     305
ccard_error_type_unknown         218
ccard_type_amex         181     183
ccard_type_bankcard              192
ccard_type_diners       175     176     177     178     179     180     182     184
ccard_type_discover              193
ccard_type_enroute               172     174
ccard_type_jcb          171     173     185
ccard_type_mastercard            187     188     189     190     191
ccard_type_name          64     320     337
ccard_type_ty            64     158     165     304
ccard_type_unknown               217     311
ccard_type_visa         186
ccard_valid             158     308
check_sum       107     113     128     132
checksum        166     235
cp              105     116     117     121
credit_card             103     116+    117     196     204     212     228     235
credit_card_in          158     204
ctype            49
dbl             106     114     122     129+
ep              276     280     281     283+    285     288
err             197     204     221     222     305     308     309     319     331
exit            334     345
fprintf         293     313     325
h       49      50     257     258     259
isdigit         146
isspace         141
j      299     302+    308     318     330     336
len             198     228     229+
len1            268     270+
len2            269     270+
length_mask              164     229
main            297
max             137     148     151
memcmp          212     270
n       59+     64      79+     80      81      85      97+     99
name             66      79      81      87      97      99
prefix          163     212+
print_version            274     301
printf          336     339     340     342     343
progname        263     291     293     317     329
s      274     280     283     288     291     306     337     338     340     341
s1              137     139     141+    143     146     150     266     268     270
s2              137     150     153     266     269     270
stderr          293     315     327
stdio           257
stdlib          258
strchr          338
string           50     259
strlen          116     212     228     268     269
strrchr         280
suffix          266     341
table           169     210+    215
table_ty        160+    161     169     195
tp              195     210+    212+    215     220     229     235
type            165     220     304     308     311     320     337
verify_checksum         103     235
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #ifndef CCARD_H
  4 | #define CCARD_H
  5 |
  6 | enum ccard_type_ty
  7 | {
  8 |       ccard_type_unknown,
  9 |       ccard_type_mastercard,
 10 |       ccard_type_visa,
 11 |       ccard_type_amex,
 12 |       ccard_type_diners,
 13 |       ccard_type_discover,
 14 |       ccard_type_enroute,
 15 |       ccard_type_jcb,
 16 |       ccard_type_bankcard
 17 | };
 18 | typedef enum ccard_type_ty ccard_type_ty;
 19 |
 20 | enum ccard_error_ty
 21 | {
 22 |       ccard_error_none = 0,
 23 |       ccard_error_type_unknown,
 24 |       ccard_error_non_numeric,
 25 |       ccard_error_too_long,
 26 |       ccard_error_length_incorrect,
 27 |       ccard_error_checksum
 28 | };
 29 | typedef enum ccard_error_ty ccard_error_ty;
 30 |
 31 |
 32 | char *ccard_type_name (ccard_type_ty);
 33 | char *ccard_error_name (ccard_error_ty);
 34 | ccard_error_ty ccard_valid (char *, ccard_type_ty *);
 35 |
 36 | #endif /* CCARD_H */
```

TEXT STATISTICS

```
   814 characters
    36 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    CCARD_H          3       4
    ccard_error_checksum            27
    ccard_error_length_incorrect            26
    ccard_error_name        33
    ccard_error_non_numeric         24
    ccard_error_none        22
    ccard_error_too_long            25
    ccard_error_ty          20      29+     33      34
    ccard_error_type_unknown        23
    ccard_type_amex         11
    ccard_type_bankcard             16
    ccard_type_diners       12
    ccard_type_discover             13
    ccard_type_enroute              14
    ccard_type_jcb          15
    ccard_type_mastercard            9
    ccard_type_name         32
    ccard_type_ty            6      18+     32      34
    ccard_type_unknown               8
    ccard_type_visa         10
    ccard_valid             34
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | #define PATCHLEVEL "1.1.D009"


TEXT STATISTICS

    75 characters
     3 lines

LEXICAL STATISTICS

     1 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    PATCHLEVEL               3
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Call undocumented Int 2Eh to invoke COMMAND.COM
  5 | **
  6 | **  demo by Bob Stout
  7 | **
  8 | **  NOTES: Dangerous code - will abort batch files in progress and
  9 | **          occasionally have other undesirable effects.
 10 | **
 11 | **          Requires INT2E.ASM
 12 | */
 13 |
 14 | #include <string.h>
 15 | #include <stdlib.h>
 16 | #include "int2e.h"
 17 |
 18 | void C_Com_Call(char *string)
 19 | {
 20 |       char *buf;
 21 |
 22 |       buf = (char *)malloc(strlen(string) + 3);
 23 |       strcat(strcpy(&buf[1], string), "\r");
 24 |       buf[0] = (char)strlen(&buf[1]);
 25 |       _Int_2E(buf);
 26 |       free(buf);
 27 | }
 28 |
 29 | #ifdef TEST
 30 |
 31 | #ifdef __WATCOMC__
 32 |  #pragma off (unreferenced);
 33 | #endif
 34 |
 35 | #ifdef __TURBOC__
 36 |  #pragma argsused
 37 | #endif
 38 |
 39 | main(int argc, char *argv[])
 40 | {
 41 |       C_Com_Call(argv[1]);
 42 |       return 0;
 43 | }
 44 |
 45 | #endif /* TEST */


TEXT STATISTICS

   819 characters
    45 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     2 constants [string]
     6 constants [numeric]

SYMBOL TABLE

    C_Com_Call              18      41
    TEST            29
    _Int_2E         25
    __TURBOC__              35
    __WATCOMC__             31
    argc            39
    argsused        36
    argv            39      41
    buf             20      22      23      24+     25      26
    free            26
    h       14      15
    main            39
    malloc          22
    off             32
    stdlib          15
    strcat          23
    strcpy          23
    string          14      18      22      23
    strlen          22      24
    unreferenced            32
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**                          CDIR.C
**
** Written By:    Lynn R. Lively
** Date Written:  9/18/91
**
** Purpose: To provide a change directory facility that will cross
**          drive/partition boundaries. Never did understand why
**          MSDOS cd wouldn't do this already.
**
**--------------------------------------------------------------------
** I hereby place this work into the Public Domain. It may be used
** for any legal purpose public or private. Use this material at
** your own risk. I accept no responsibility for the accuracy or
** usability of the information contained herein. Neither do I
** accept liability for any possible damage caused by use of this
** material. However, should you have a problem, question, or
** suggestion I would be glad to help in any way that I can. You
** can reach me at (H) 713-893-7875 or (W) 713-591-6611 x 149.
**--------------------------------------------------------------------
*/

/*
**                          Change History
**
**   Rev #   Date       By      Description of change
**   1.00  | 09/18/91 | LRL | Original Version
**   1.01  | 09/18/91 | RBS | Added MSC, ZTC support for SNIPPETS
**--------------------------------------------------------------------
** Directory of initials:
** Initials          Name
** LRL        Lynn R. Lively
** RBS        Bob Stout
*/


#include <stdio.h>
#include <string.h>

#if defined(__TURBOC__) || defined(__POWERC)
 #include <dir.h>
#else
 #include <dos.h>
 #include <direct.h>

 #ifdef __ZTC__
  #define _dos_getdrive(d) dos_getdrive(d)
  #define _dos_setdrive(d,m) dos_setdrive(d,m)
 #endif

 #if defined(__ZTC__) || defined(__WATCOMC__)
  #define drive_t unsigned
 #else
  #define drive_t int
 #endif

 #if defined(_MSC_VER) || defined(__WATCOMC__)
 drive_t getdisk(void)
 {
       drive_t drive;

       _dos_getdrive(&drive);
       return drive - 1;
 }

 drive_t setdisk(drive_t drive)
 {
       drive_t max_drives;

       _dos_setdrive(drive + 1, &max_drives);
       return max_drives - 1;
 }
 #endif
#endif

main (int argc, char * argv[])
{
      int d;
      int max_d;

      char wk_str[128];

      if (argc > 1)
      {
            strupr (argv[1]);
            if (argv[1][1] == ':')
            {
                  /*
                  ** Find out what the maximum drive number can be.
                  */

                  max_d = getdisk ();
                  max_d = setdisk (max_d);

                  d = argv[1][0] - 'A';
                  if (d < max_d)
                  {
                        /*
```

```
101 |                           ** If the drive specification was valid position
102 |                           ** to it and then do a change directory.
103 |                           */
104 |
105 |                           setdisk (d);
106 |                           chdir (argv[1]);
107 |                     }
108 |                     else
109 |                     {
110 |                           puts ("Invalid drive specification");
111 |                           return -1;
112 |                     }
113 |               }
114 |               else
115 |               {
116 |                     /*
117 |                     ** If the argument has no drive spec just do a regular
118 |                     ** change directory.
119 |                     */
120 |
121 |                     chdir (argv[1]);
122 |               }
123 |         }
124 |         else
125 |         {
126 |               /*
127 |               ** If no arguments are passed, return the current working
128 |               ** directory path just like MSDOS cd does.
129 |               */
130 |
131 |               puts (getcwd (wk_str, sizeof (wk_str)));
132 |         }
133 |         return 0;
134 | }
```

TEXT STATISTICS

```
  3640 characters
   134 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    19 preprocessor instructions
     2 constants [character]
     1 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

```
    _MSC_VER        59
    __POWERC        42
    __TURBOC__              42
    __WATCOMC__             53      59
    __ZTC__         48      53
    _dos_getdrive           49      64
    _dos_setdrive           50      72
    argc            78      85
    argv            78      87      88      97     106     121
    chdir          106     121
    d       49+     50+     80      97      98     105
    defined         42+     53+     59+
    dir             43
    direct          46
    dos             45
    dos_getdrive            49
    dos_setdrive            50
    drive           62      64      65      68      72
    drive_t         54      56      60      62      68+     70
    getcwd         131
    getdisk         60      94
    h       39      40      43      45      46
    m       50+
    main            78
    max_d           81      94      95+     98
    max_drives              70      72      73
    puts           110     131
    setdisk         68      95     105
    stdio           39
    string          40
    strupr          87
    wk_str          83     131+
```

```
/* +++Date last modified: 05-Jul-1997 */

/* public domain by Jerry Coffin.  Tested with MSC 7.0  */
/* written primarily for clarity, not speed...          */
/* requires w_wrap.c and w_wrap.h from snippets.        */


#include <conio.h>
#include <string.h>
#include "w_wrap.h"

void center(FILE *file, char *string, size_t width)
{
      char *line,*end_line;
      size_t spaces;
      int last = 0;
      size_t len;

      word_wrap(string,width);
      line = string;
      while (!last)
      {
            end_line = strchr(line,'\n');
            if (NULL==end_line)
            {
                  last = 1;
                  end_line = strchr(line,'\0');
            }
            len = end_line - line;
            spaces = (width - len) / 2 ;
            fprintf(file,"\n%*c%*.*s",spaces,' ',len,len,line);
            line = end_line + 1;
      }
}

#ifdef TEST

int main(void)
{
      char *string = "This is a long string to see if it will be"
            " printed out correctly but I'm not sure whether it will work"
            " correctly or not.  I guess we'll see when we try it out.";

      printf("\nHere's the string centered in 50 columns.\n");
      center(stdout,string,50);
      printf("\n\nAnd here it's centered in 72 columns.\n");
      center(stdout,string,72);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1386 characters
    51 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     3 constants [character]
     7 constants [string]
     7 constants [numeric]

SYMBOL TABLE

    FILE            12
    NULL            24
    TEST            36
    center          12      45      47
    conio            8
    end_line        14      23      24      27      29      32
    file            12      31
    fprintf         31
    h        8       9
    last            16      21      26
    len             17      29      30      31+
    line            14      20      23      27      29      31      32
    main            38
    printf          44      46
    size_t          12      15      17
    spaces          15      30      31
    stdout          45      47
    strchr          23      27
    string           9      12      19      20      40      45      47
    width           12      19      30
    word_wrap       19
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    CFG.c       Generic configuration file handler.

                A. Reitsma, Delft, The Netherlands.
                v1.00  94-07-09  Public Domain.
 ---------------------------------------------------------------------- */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "cfg.h"

#define LINE_LEN_MAX    128                 /* actual max line length  */
#define BUFFERSIZE      LINE_LEN_MAX +2     /* ... including \n and \0 */

enum RetVal
{
    NO_PROBLEMS,
    ERR_FOPEN,
    ERR_MEM,
};

int CfgRead( char * Filename, struct CfgStrings * CfgInfo )
{
    char Buffer[ BUFFERSIZE ];
    char * WorkPtr ;
    char * CfgName ;
    char * CfgData ;
    struct CfgStrings * Cfg ;
    FILE * CfgFile ;

    CfgFile = fopen( Filename, "r" );
    if( NULL == CfgFile )
    {
        return ERR_FOPEN ;
    }

    while( NULL != fgets( Buffer, BUFFERSIZE, CfgFile ))
    {
        /* clip off optional comment tail indicated by a semi-colon
        */
        if( NULL != (WorkPtr = strchr( Buffer, ';' )))
            *WorkPtr = '\0';
        else
            WorkPtr = Buffer + strlen( Buffer );

        /* clip off trailing and leading white space
        */
        WorkPtr--;
        while( isspace( *WorkPtr ) && WorkPtr >= Buffer )
            *WorkPtr-- = '\0';
        WorkPtr = Buffer;
        while( isspace( *WorkPtr ))
            WorkPtr++;
        if( 0 == strlen( WorkPtr ))
            continue;

        CfgName = strtok( WorkPtr, " =" );
        if( NULL != CfgName )
        {
            /* Condition the name (lower case required),
               and strip leading white and a 'late' = from data part.
            */
            strlwr( CfgName );
            CfgData = strtok( NULL, "" );
            while( isspace( *CfgData ))
                CfgData++;
            if( '=' == *CfgData )
                CfgData++;
            while( isspace( *CfgData ))
                CfgData++;

            /* look for matching 'name'
            */
            Cfg = CfgInfo ;
            while( NULL != Cfg->name && 0 != strcmp( Cfg->name, CfgName ))
                Cfg++;

            /* duplicate the data if the name is found.
            */
            if( NULL != Cfg->name )
            {
                Cfg->data = strdup( CfgData ); /* strdup is not ANSI    */
                                               /* memory leaks if Cfg->data */
                                               /* is malloc'ed already      */
                if( NULL == Cfg->data )
                {
                    fclose( CfgFile );
                    return ERR_MEM;
                }
            }   /* undetected error on failure should not be a problem  */
                /* as configuration reading should be done early.       */
        }       /* but test and handle it anyway ...                    */
    }
    fclose( CfgFile );
    return NO_PROBLEMS ;
}
```

```
 101 | #ifdef TEST
 102 |
 103 | struct CfgStrings CfgDemoData[] =
 104 | {
 105 |     { "workpath", NULL },
 106 |     { "archiver", NULL },
 107 |     { "splitter", NULL },
 108 |     { NULL, NULL }         /* array terminator. REQUIRED !!! */
 109 | };
 110 |
 111 | main( int argc, char * argv[] )
 112 | {
 113 |     int ix ;
 114 |
 115 |     if( argc > 1 )
 116 |         CfgRead( argv[ 1 ], CfgDemoData );
 117 |     else
 118 |         return ERR_FOPEN ;
 119 |
 120 |     printf( "%s\n", argv[ 0 ]);
 121 |     for( ix = 0; ix < 4 ; ix++ ) /* intentionally one too much */
 122 |        printf( "CfgItem \"%s\" is \"%s\"\n", CfgDemoData[ ix ].name,
 123 |                                              CfgDemoData[ ix ].data );
 124 |     return NO_PROBLEMS;
 125 | }
 126 | #endif
 127 |
 128 | /* ==== CFG.c end ================================================== */
```

TEXT STATISTICS

```
  3825 characters
   128 lines
```

LEXICAL STATISTICS

```
    18 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     4 constants [character]
     9 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    BUFFERSIZE              16      27      40
    Buffer          27      40      44      47+     52      54
    Cfg             31      77      78+     79      83      85      88
    CfgData         30      67      68      69      70      71      72      73      85
    CfgDemoData             103     116     122     123
    CfgFile         32      34      35      40      90      97
    CfgInfo         25      77
    CfgName         29      60      61      66      78
    CfgRead         25      116
    CfgStrings              25      31      103
    ERR_FOPEN       21      37      118
    ERR_MEM         22      91
    FILE            32
    Filename        25      34
    LINE_LEN_MAX            15      16
    NO_PROBLEMS             20      98      124
    NULL            35      40      44      61      67      78      83      88      105     106     107
          108+
    RetVal           18
    TEST            101
    WorkPtr          28      44      45      47      51      52+     53      54      55      56      57
           60
    argc            111     115
    argv            111     116     120
    ctype            12
    data             85      88      123
    fclose           90      97
    fgets            40
    fopen            34
    h       10       11      12
    isspace          52      55      68      72
    ix              113     121+    122     123
    main            111
    name             78+     83      122
    printf          120     122
    stdio            10
    strchr           44
    strcmp           78
    strdup           85
    string           11
    strlen           47      57
    strlwr           66
    strtok           60      67
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /* ======================================================================
   4 |     CFG.h       Configuration file handler.
   5 |                 A. Reitsma, Delft, The Netherlands.
   6 |                 v1.00  94-07-09  Public Domain.
   7 |  ---------------------------------------------------------------------- */
   8 |
   9 | struct CfgStrings
  10 | {
  11 |     char * name ;
  12 |     char * data ;
  13 | };
  14 |
  15 | int CfgRead( char * Filename, struct CfgStrings * CfgInfo );
  16 |
  17 | /* ==== CFG.h end ================================================== */
```

TEXT STATISTICS

```
   551 characters
    17 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CfgInfo         15
    CfgRead         15
    CfgStrings              9      15
    Filename        15
    data            12
    name            11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  change_prn()
**
**  A function to change the standard printer device for the duration
**  of a program. Valid new device codes are:
**
**      0 - LPT1
**      1 - LPT2
**      2 - LPT3
**      3 - COM1
**      4 - COM2
**      5 - CON
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include "sniprint.h"               /* Includes PrintDevice typedef     */

int change_prn(PrintDevice device)
{
      char *newdev;

      switch (device)
      {
      case LPT1:
            newdev = "LPT1";
            break;
      case LPT2:
            newdev = "LPT2";
            break;
      case LPT3:
            newdev = "LPT3";
            break;
      case COM1:
            newdev = "COM1";
            break;
      case COM2:
            newdev = "COM2";
            break;
      case CON:
            newdev = "CON";
            break;
      default:
            return -1;
      }
      return (NULL == freopen(newdev, "w", stdprn));
}
```

```
TEXT STATISTICS

  1296 characters
    54 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     8 constants [string]
     1 constants [numeric]

SYMBOL TABLE

    COM1            41
    COM2            44
    CON             47
    LPT1            32
    LPT2            35
    LPT3            38
    NULL            53
    PrintDevice             26
    change_prn              26
    device          26      30
    freopen         53
    newdev          28      33      36      39      42      45      48      53
    stdprn          53
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  CHBYTES.C - Change bytes in a file
**
**  This program searches a file for a sequence of bytes. If they are
**  found, they are replaced with zeros. It was originally developed for
**  a friend who needed a program to call from Paradox to remove printer
**  control sequences from formatted print files. The requirements were
**  1) since it is called by another program, all status has to be returned
**  in the errorlevel with no screen messages allowed, and 2) The file must
**  remain the same length, so the deleted sequences must only be replaced
**  with ASCII NULs.
**
**  Syntax: CHBYTES filename pattern_1 [pattern_2 [...pattern_N]]
**  where:  Each pattern is a comma-separated list of bytes, each of which
**          may be of the following forms:
**          C    - Any single character will be treated as literal.
**          XXh  - "XX" will be interpreted as a hexadecimal number (both
**                 1- and 2-digit hex numbers are allowed).
**          NNNd - "NNN" will be interpreted as a decimal number (both
**                 1-, 2-, and 3-digit decimal numbers are allowed).
**
**  e.g.    CHBYTES printer.fil 12d 1bh,[,3,x
**          would zero out form feeds and the escape sequence "[3x"
**
**  Returns: 0 - Success
**           1 - No filename
**           2 - No arguments
**           3 - Error opening file
**           4 - Not enough memory
**           5 - Bad argument
**           6 - Error reading file
**           7 - Error writing file
**
**  Public domain by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "sniptype.h"

#ifdef __ZTC__
 int _okbigbuf = 0;
#endif

#ifndef max
 #define max(x,y) (((x) >= (y)) ? (x) : (y))
#endif

int bufsize;

struct {
      char pattern[40];                         /* pattern to find      */
      int numbytes;                             /* length of pattern    */
} search[40];

int main (int argc, char *argv[])
{
      FILE *fp  = NULL;
      char *buf = NULL, *getbuf(void);
      fpos_t rpos;
      int i, patterns, max_bytes = 0;
      Boolean_T hex2char(const char *, char *);

      if (2 > argc)                             /* no filename          */
            return 1;
      if (3 > argc)                             /* no argument          */
            return 2;
      if (NULL == (fp = fopen(argv[1], "r+b")))
            return 3;                           /* file open error      */
      if (NULL == (buf = getbuf()))
            return 4;                           /* no memory for buffer */

      patterns = argc - 2;                      /* process arguments    */
      for (i = 2; i < argc; ++i)
      {
            char *p, *ptr;

            if (NULL != (ptr = strtok(argv[i], ",")))
            {
                  p = search[i - 2].pattern;
                  do
                  {
                        search[i - 2].numbytes++;
                        if (1 == strlen(ptr))
                        {
                              *p++ = *ptr;
                              continue;
                        }
                        switch (toupper(LAST_CHAR(ptr)))
                        {
                        case 'D':
                              LAST_CHAR(ptr) = '\0';
                              *p++ = (char)atoi(ptr);
                              break;
                        case 'H':
                              LAST_CHAR(ptr) = '\0';
```

```
                                if (Error_ == hex2char(ptr, p++))
                                      return 5;
                                break;
                          default:
                                return 5;
                          }
                    } while (NULL != (ptr = strtok(NULL, ",")));
                    *p = '\0';
                    max_bytes = max(max_bytes, search[i - 2].numbytes);
              }
              else  return 5;
        }

        fgetpos(fp, &rpos);                       /* save where we are    */
        while (1)
        {
              int bytes, n;
              Boolean_T modified;

              if (max_bytes > (bytes = (int)fread(buf, 1, bufsize, fp)))
              {
                    if (0 == bytes && !feof(fp))
                          return 6;               /* something's wrong!   */
                    else  break;                  /* all done!            */
              }
              for (n = 0, modified = False_; n < patterns; ++n)
              {
                    /* check each pattern in turn                        */

                    for (i = 0; i < (bytes - max_bytes + 1); ++i)
                    {
                          int j;

                          if (buf[i] != *(search[n].pattern))
                                continue;
                          if (Success_ != strncmp(&buf[i],
                                search[n].pattern, search[n].numbytes))
                          {
                                continue;
                          }

                          /* found one! replace it in the buffer          */

                          for (j = 0; j < search[n].numbytes; ++j, ++i)
                                buf[i] = '\0';
                          modified = True_;
                    }
              }
              if (modified)                       /* write changes, if any*/
              {
                    fpos_t wpos = rpos;

                    fsetpos(fp, &wpos);
                    if (bytes != (int)fwrite(buf, 1, bytes, fp))
                          return 7;
                    fsetpos(fp, &rpos);
              }
              rpos += bytes - max_bytes + 1;      /* get another buffer   */
              fsetpos(fp, &rpos);
        }
        fclose(fp);
        return Success_;
}

/*
**  Allocate the largest buffer we can
*/

char *getbuf(void)
{
        register char *buffer;

        for (bufsize = 0x4000; bufsize >= 128; bufsize >>= 1)
        {
              if (NULL != (buffer = (char *) malloc(bufsize)))
                    return buffer;
        }
        return NULL;
}

/*
**  Convert ASCII hex char to char
*/

#define xdigit(c) (toupper(c) - (((c) > '9') ? 'A' - 10 : '0'))

Boolean_T hex2char(const char *hex, char *buf)
{
        int ch = 0;
        char *p = (char *)hex;

        while(*p)
        {
              if (!isxdigit(*p))
                    return Error_;
              ch <<= 4;
              ch  += xdigit(*p);
                  ++p;
        }
        *buf = (char)ch;
```

```
 201 |       return Success_;
 202 | }
```

TEXT STATISTICS

```
  6630 characters
   202 lines
```

LEXICAL STATISTICS

```
    18 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     9 constants [character]
     4 constants [string]
    37 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       66     118     187
    Error_         101     195
    FILE            62
    False_         126
    LAST_CHAR       93      96     100
    NULL            62      63      72      74      82     107+    175     178
    Success_       136     162     201
    True_          146
    __ZTC__         45
    _okbigbuf       46
    argc            60      68      70      77      78
    argv            60      72      82
    atoi            97
    buf             63      74     120     134     136     145     154     187     200
    buffer         171     175     176
    bufsize         53     120     173+    175
    bytes          117     120     122     130     154+    158
    c      185+
    ch             189     196     197     200
    ctype           42
    fclose         161
    feof           122
    fgetpos        114
    fopen           72
    fp              62      72     114     120     122     153     154     156     159     161
    fpos_t          64     151
    fread          120
    fsetpos        153     156     159
    fwrite         154
    getbuf          63      74     169
    h       39      40      41      42
    hex            187     190
    hex2char        66     101     187
    i       65      78+     82      84      87     109     130+    134     136     144     145
    isxdigit       194
    j      132     144+
    main            60
    malloc         175
    max             49      50     109
    max_bytes       65     109+    120     130     158
    modified       118     126     146     149
    n      117     126+    134     137+    144
    numbytes        57      87     109     137     144
    p       80      84      90      97     101     108     190     192     194     197     198
    pattern         56      84     134     137
    patterns        65      77     126
    ptr             80      82      88      90      93      96      97     100     101     107
    rpos            64     114     151     156     158     159
    search          58      84      87     109     134     137+    144
    stdio           39
    stdlib          40
    string          41
    strlen          88
    strncmp        136
    strtok          82     107
    toupper         93     185
    wpos           151     153
    x       50+
    xdigit         185     197
    y       50+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** CHECKEXE.C - checksum protection for executable files
**
** by: Bob Jarvis
*/

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "crc.h"

static struct {
      unsigned char marker[16];
      unsigned long checksum;
} marker_struct = {"CHECKEXE MARKER",0L};

void checkexe(char *fname)
{
      FILE *fptr;
      unsigned int c;
      int first_time = 0, i;
      char buffer[14];
      unsigned long chksum = 0L;
      unsigned long l;
      unsigned long marker_offset;
      unsigned char *charptr;
      time_t tm;

      fptr = fopen(fname,"r+b");
      if(fptr == NULL)
      {
            fprintf(stderr,"checkexe : unable to open input file '%s'\n",
                  fname);
            exit(99);
      }

      setvbuf(fptr, NULL, _IOFBF, 32767);    /* try to get a 32K buffer */

   /*
    * If this is the first time the check has been run, scan the entire file
    * to find the marker.  Otherwise proceed.
    */

      if(marker_struct.checksum == 0L)
      {
            first_time = 1;

            c = fgetc(fptr);
            while(!feof(fptr))
            {
                  if(c == (unsigned int)marker_struct.marker[0])
                  {
                        fread(buffer,sizeof(buffer),1,fptr);
                        if(memcmp(buffer,&marker_struct.marker[1],
                              sizeof(buffer)) == 0)
                        {
                              marker_offset = ftell(fptr) + 1L;
                              break;
                        }
                        fseek(fptr,-13L,SEEK_CUR);
                  }

                  c = fgetc(fptr);
            }

            if(feof(fptr))
            {
                  fprintf(stderr,"checkexe : unable to locate marker\n");
                  exit(99);
            }

      /* Change the marker field to random values */

            tm = time(NULL);

            srand((unsigned int)tm);

            for(i = 0 ; i < sizeof(marker_struct.marker) ; ++i)
                  marker_struct.marker[i] = (unsigned char) rand();

            fseek(fptr,marker_offset - sizeof(marker_struct.marker),SEEK_SET);

            fwrite(marker_struct.marker,sizeof(marker_struct.marker),1,fptr);
      }

   /* Calculate the checksum for the entire file */

      rewind(fptr);

      c = fgetc(fptr);
      for(l = 0 ; !feof(fptr) ; ++l)
      {
            chksum += (unsigned long)c;
            c = fgetc(fptr);
      }

      if(first_time)
```

```
101 |       {
102 |             marker_struct.checksum = chksum;
103 |             fseek(fptr,marker_offset,SEEK_SET);
104 |             fwrite(&marker_struct.checksum,sizeof(unsigned long),1,fptr);
105 |       }
106 |       else
107 |       {
108 |             charptr = (unsigned char*) &marker_struct.checksum;
109 |
110 |             for(i = 0 ; i < sizeof(marker_struct.checksum) ; ++i)
111 |                   chksum -= (unsigned long)(charptr[i]);
112 |
113 |             if(chksum != marker_struct.checksum)
114 |             {
115 |                   fprintf(stderr, "\acheckexe : %s has been altered, "
116 |                         "possibly by a virus\n", fname);
117 |                   exit(99);
118 |             }
119 |       }
120 |
121 |       fclose(fptr);
122 |       return;
123 | }
124 |
125 | #ifdef TEST
126 |
127 | #ifdef __WATCOMC__
128 |  #pragma off (unreferenced);
129 | #endif
130 | #ifdef __TURBOC__
131 |  #pragma argsused
132 | #endif
133 |
134 | main(int argc, char *argv[])
135 | {
136 |       checkexe(argv[0]);
137 |       return EXIT_SUCCESS;
138 | }
139 |
140 | #endif /* TEST */
```

TEXT STATISTICS

```
  3549 characters
   140 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     7 constants [string]
    23 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS           137
    FILE            22
    NULL            33      40      77
    SEEK_CUR        63
    SEEK_SET        84     103
    TEST           125
    _IOFBF          40
    __TURBOC__             130
    __WATCOMC__            127
    argc           134
    argsused       131
    argv           134     136
    buffer          25      56+     57      58
    c       23      51      54      66      93      96      97
    charptr         29     108     111
    checkexe        20     136
    checksum        17      47     102     104     108     110     113
    chksum          26      96     102     111     113
    exit            37      72     117
    fclose         121
    feof            52      69      94
    fgetc           51      66      93      97
    first_time              24      49     100
    fname           20      32      36     116
    fopen           32
    fprintf         35      71     115
    fptr            22      32      33      40      51      52      56      60      63      66      69
            84      86      91      93      94      97     103     104     121
    fread           56
    fseek           63      84     103
    ftell           60
    fwrite          86     104
    h        9      10      11      12
    i       24      81+     82     110+    111
    l       27      94+
    main           134
    marker          16      54      57      81      82      84      86+
    marker_offset           28      60      84     103
    marker_struct           18      47      54      57      81      82      84      86+    102     104
           108     110     113
    memcmp          57
    off            128
    rand            82
    rewind          91
    setvbuf         40
    srand           79
    stderr          35      71     115
    stdio            9
    stdlib          12
    string          10
    time            11      77
    time_t          30
    tm              30      77      79
    unreferenced           128
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  CHECKSUM.C - Compute the checksum of a file
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include "crc.h"
 10 |
 11 | unsigned checksum(void *buffer, size_t len, unsigned int seed)
 12 | {
 13 |       unsigned char *buf = (unsigned char *)buffer;
 14 |       size_t i;
 15 |
 16 |       for (i = 0; i < len; ++i)
 17 |             seed += (unsigned int)(*buf++);
 18 |       return seed;
 19 | }
 20 |
 21 | #ifdef TEST
 22 |
 23 | #include <stdio.h>
 24 |
 25 | main()
 26 | {
 27 |       FILE *fp;
 28 |       size_t len;
 29 |       char buf[4096], *file = "CHECKSUM.C";
 30 |
 31 |       if (NULL == (fp = fopen(file, "rb")))
 32 |       {
 33 |             printf("Unable to open %s for reading\n", file);
 34 |             return -1;
 35 |       }
 36 |       len = fread(buf, sizeof(char), sizeof(buf), fp);
 37 |       printf("%d bytes read\n", len);
 38 |       printf("The checksum of %s is %#x\n", file, checksum(buf, len, 0));
 39 |       return 0;
 40 | }
 41 |
 42 | #endif /* TEST */
```

TEXT STATISTICS

```
   897 characters
    42 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     6 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    FILE            27
    NULL            31
    TEST            21
    buf             13      17      29      36+     38
    buffer          11      13
    checksum        11      38
    file            29      31      33      38
    fopen           31
    fp              27      31      36
    fread           36
    h       23
    i       14      16+
    len             11      16      28      36      37      38
    main            25
    printf          33      37      38
    seed            11      17      18
    size_t          11      14      28
    stdio           23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  chgext.c - Change a file's extension
**
**  public domain by Bob Stout
**
**  Arguments: 1 - Pathname
**             2 - Old extension (NULL if don't care)
**             3 - New extension
**
**  Returns: Pathname or NULL if failed
**
**  Note: Pathname buffer must be long enough to append new extension
**
**  Side effect: Converts Unix style pathnames to DOS style
*/

#include <stdio.h>
#include <string.h>
#include "filnames.h"

char *chgext(char *path, char *oldext, char *newext)
{
      char *p;

      /* Convert to DOS-style path name */

      for (p = path; *p; ++p)
            if ('/' == *p)
                  *p = '\\';

      /* Find extension or point to end for appending */

      if (NULL == (p = strrchr(path, '.')) || NULL != strchr(p, '\\'))
            p = strcpy(&path[strlen(path)], ".");
      ++p;

      /* Check for old extension */

      if (oldext && strcmp(p, oldext))
            return NULL;

      /* Add new extension */

      while ('.' == *newext)
            ++newext;
      strncpy(p, newext, 3);

      /*
      ** Added to insure string is properly terminated. Without this, if
      ** the new extension is longer than the old, we lose the terminator.
      */

      p[strlen(newext)] = '\0';

      return path;
}

#ifdef TEST

main(int argc, char *argv[])
{
      char *retval, *old_ext = NULL, path[128];

      if (2 > argc)
      {
            puts("Usage: CHGEXT path [old_ext]");
            puts("\nChanges extension to \".TST\"");
            puts("Old extension optional");
            return -1;
      }
      strcpy(path, strupr(argv[1]));
      if (2 < argc)
            old_ext = strupr(argv[2]);
      if (NULL == (retval = chgext(path, old_ext, ".TSTstuff")))
            puts("chgext() failed");
      else  printf("chgext(%s, %s, TST)\n...returned...\n%s\n", argv[1],
            old_ext ? old_ext : "NULL", path);
      return (NULL == retval);
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2064 characters
    83 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     6 constants [character]
     9 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

NULL		35+	42	64	76	80				
TEST		60								
argc		62	66	74						
argv		62	73	75	78					
chgext		23	76							
h	19	20								
main		62								
newext		23	46	47	48	55				
old_ext		64	75	76	79+					
oldext		23	41+							
p	25	29+	30	31	35+	36	37	41	48	55
path		23	29	35	36+	57	64	73	76	79
printf		78								
puts		68	69	70	77					
retval		64	76	80						
stdio		19								
strchr		35								
strcmp		41								
strcpy		36	73							
string		20								
strlen		36	55							
strncpy		48								
strrchr		35								
strupr		73	75							

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************************

    CHKREG.C - Validates a key created by REGIT.C

   Donated to the Public Domain by Craig Morrison 12 May 1994, use,
   abuse, fold, spindle or mutilate anyway you see fit.

*************************************************************************/

#include "regkey.h"

/*************************************************************************

    CHKREG accepts two arguments on its command line; The key value in
    hexadecimal generated by REGIT and a string. You should end up with
    XOR_PRIME after the XOR manipulations, if not, then the given key
    was invalid.

*************************************************************************/

int main(int argc, char *argv[])
{
      long keyval;
      long key;
      char *p;
      char buf[128];

      if (argc>2)
      {
            strcpy(buf, argv[1]);
            strupr(buf);
            sscanf(buf, "%8lX", &keyval);
            keyval ^= XOR_POST_CRYPT;

            strcpy(buf, argv[2]);
            p = strrev(buf);
            while(*p)
            {
                  if (*p=='_')
                        *p = ' ';

                  key = (long) toupper(*p);
                  key ^= (long)XOR_CRYPT;
                  keyval ^= key;
                  p++;
            }
            printf("Key value = %08lX hex.\n", keyval);
      }
      return 0;
}
```

```
TEXT STATISTICS

  1502 characters
    52 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     2 constants [character]
     3 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    XOR_CRYPT       45
    XOR_POST_CRYPT          35
    argc            23      30
    argv            23      32      37
    buf             28      32      33      34      37      38
    key             26      44      45      46
    keyval          25      34      35      46      49
    main            23
    p       27      38      39      41      42      44      47
    printf          49
    sscanf          34
    strcpy          32      37
    strrev          38
    strupr          33
    toupper         44
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  CHMOD.C - Retrieve or change a DOS file's attributes
**
**  public domain demo by Bob Stout
**
**  Notes: To expand command line arguments with wildcards,
**         TC/TC++/BC++  - Link in WILDARGS.OBJ.
**         MSC/QC        - Link in SETARGV.OBJ.
**         ZTC/C++       - Link in _MAINx.OBJ, where 'x' is the memory model.
**
**         Allows file list(s) using standard "@file_list_name" convention.
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <ctype.h>
#include <string.h>

#define LAST_CHAR(s) (s)[strlen(s)-1]

#if defined(__TURBOC__)
 #include <io.h>
 #define FAR far
#else
 #include <stdarg.h>

 #define TOBOOL(x) (!(!(x)))
 #define FAR _far

 #if (defined(_MSC_VER) && (_MSC_VER >= 700)) || (defined(__SC__))
  /* Make FP_xxx macros lvalues as in older versions */
  #undef FP_SEG
  #undef FP_OFF
  #define FP_SEG(fp)    ((unsigned)((unsigned long)(fp) >> 16))
  #define FP_OFF(fp)    ((unsigned)(fp && 0xffff))
 #endif

 int _chmod(const char *path, int func, ...)
 {
       union REGS regs;
       struct SREGS sregs;
       int atr = 0;
       va_list args;

       if (0 != (func = TOBOOL(func)))
       {
            va_start(args, func);
            atr = va_arg(args, int);
       }
       regs.x.ax = 0x4300 + func;
       regs.x.dx = FP_OFF((char FAR *)path);
       regs.x.cx = atr;
       segread(&sregs);
       sregs.ds  = FP_SEG((char FAR *)path);
       intdosx(&regs, &regs, &sregs);
       if (regs.x.cflag)
            return -1;
       if (func)
            return atr;
       else return regs.x.cx;
 }

 #ifndef FA_RDONLY
  #define FA_RDONLY _A_RDONLY
 #endif

 #ifndef FA_HIDDEN
  #define FA_HIDDEN _A_HIDDEN
 #endif

 #ifndef FA_SYSTEM
  #define FA_SYSTEM _A_SYSTEM
 #endif

 #ifndef FA_ARCH
  #define FA_ARCH   _A_ARCH
 #endif

 #ifndef FA_LABEL
  #define FA_LABEL  _A_VOLID
 #endif

 #ifndef FA_DIREC
  #define FA_DIREC  _A_SUBDIR
 #endif
#endif

int attrib,                   /* Set up new attributes here             */
    atr_setmask = 0,
    atr_clrmask = -1,
    flag = 0;                 /* Passed as func to _chmod()             */

void usage(void)              /* Tell 'em they messed up                */
{
      puts("Usage: CHMOD file [file [...file] [+switches] [-switches]");
      puts("Where switches are one or more of:");
      puts("    A: Archive");
```

```
101 |       puts("    R: Read only");
102 |       puts("    H: Hidden");
103 |       puts("    S: System");
104 |       puts("File lists may be specified with \"@file_list_name\"");
105 |       puts("With no switches, diplays current attributes.");
106 |       puts("Displayed attributes are as above plus:");
107 |       puts("    D: Subdirectory");
108 |       puts("    V: Volume label");
109 |       exit(1);
110 | }
111 |
112 | void setattr(char atr)        /* Set bits in attribute                */
113 | {
114 |       switch (toupper(atr))
115 |       {
116 |       case 'A':
117 |             atr_setmask |= FA_ARCH;
118 |             break;
119 |       case 'R':
120 |             atr_setmask |= FA_RDONLY;
121 |             break;
122 |       case 'S':
123 |             atr_setmask |= FA_SYSTEM;
124 |             break;
125 |       case 'H':
126 |             atr_setmask |= FA_HIDDEN;
127 |             break;
128 |       default:
129 |             usage();
130 |       }
131 | }
132 |
133 | void clrattr(char atr)        /* Clear bits in attribute              */
134 | {
135 |       switch (toupper(atr))
136 |       {
137 |       case 'A':
138 |             atr_clrmask &= ~FA_ARCH;
139 |             break;
140 |       case 'R':
141 |             atr_clrmask &= ~FA_RDONLY;
142 |             break;
143 |       case 'S':
144 |             atr_clrmask &= ~FA_SYSTEM;
145 |             break;
146 |       case 'H':
147 |             atr_clrmask &= ~FA_HIDDEN;
148 |             break;
149 |       default:
150 |             usage();
151 |       }
152 | }
153 |
154 | void show_atr(char *path)
155 | {
156 |       char astr[7], *ptr;
157 |
158 |       if (-1 == (attrib = _chmod(strupr(path), 0)))
159 |       {
160 | ATR_ERR:    printf("\aCHMOD: Error! (file: %s)", path);
161 |             exit(-1);
162 |       }
163 |       attrib |= atr_setmask;
164 |       attrib &= atr_clrmask;
165 |       if (-1 == (attrib = _chmod(path, flag, attrib)))
166 |             goto ATR_ERR;
167 |       ptr = astr;
168 |       *ptr++ = (char)((attrib & FA_ARCH)   ? 'A' : '.');
169 |       *ptr++ = (char)((attrib & FA_RDONLY) ? 'R' : '.');
170 |       *ptr++ = (char)((attrib & FA_SYSTEM) ? 'S' : '.');
171 |       *ptr++ = (char)((attrib & FA_HIDDEN) ? 'H' : '.');
172 |       *ptr++ = (char)((attrib & FA_DIREC)  ? 'D' : '.');
173 |       *ptr++ = (char)((attrib & FA_LABEL)  ? 'V' : '.');
174 |       *ptr = '\0';
175 |       printf("%-15s %s\n", path, astr);
176 | }
177 |
178 | int main (int argc, char *argv[])
179 | {
180 |       int i, j;
181 |
182 |       if (2 > argc)
183 |             usage();
184 |       for (i = 1; i < argc; ++i)    /* Build attribute masks          */
185 |       {
186 |             switch (*argv[i])
187 |             {
188 |             case '+':
189 |                   for (j = 1; argv[i][j]; ++j)
190 |                         setattr(argv[i][j]);
191 |                   flag = 1;
192 |                   break;
193 |             case '-':
194 |                   for (j = 1; argv[i][j]; ++j)
195 |                         clrattr(argv[i][j]);
196 |                   flag = 1;
197 |                   break;
198 |             default:
199 |                   break;            /* Assume it's a file name        */
200 |             }
```

```
201 |       }
202 |       for (i = 1; i < argc; ++i)    /* Scan filenames                   */
203 |       {
204 |             if (strchr("+-", *argv[i]))
205 |                   continue;
206 |             if ('@' == *argv[i])
207 |             {
208 |                   FILE *fp;
209 |                   char buf[256], *ptr = &argv[i][1];
210 |
211 |                   if (NULL == (fp = fopen(ptr, "r")))
212 |                   {
213 |                         printf("\aCHMOD: Error opening %s\n", ptr);
214 |                         return -1;
215 |                   }
216 |                   while (NULL != fgets(buf, 255, fp))
217 |                   {
218 |                         while ('\n' == LAST_CHAR(buf))     /* Strip '\n'*/
219 |                               LAST_CHAR(buf) = '\0';
220 |                         show_atr(buf);
221 |                   }
222 |                   fclose(fp);
223 |             }
224 |             else  show_atr(argv[i]);
225 |       }
226 |       return 0;
227 | }
```

TEXT STATISTICS

6157 characters
227 lines

LEXICAL STATISTICS

12 comments [std-C]
0 comments [C++]
38 preprocessor instructions
26 constants [character]
16 constants [string]
29 constants [numeric]

SYMBOL TABLE

```
ATR_ERR          160     166
FAR               26      31      54      57
FA_ARCH           78      79     117     138     168
FA_DIREC          86      87     172
FA_HIDDEN         70      71     126     147     171
FA_LABEL          82      83     173
FA_RDONLY         66      67     120     141     169
FA_SYSTEM         74      75     123     144     170
FILE             208
FP_OFF            36      38      54
FP_SEG            35      37      57
LAST_CHAR         22     218     219
NULL             211     216
REGS              43
SREGS             44
TOBOOL            30      48
_A_ARCH           79
_A_HIDDEN         71
_A_RDONLY         67
_A_SUBDIR         87
_A_SYSTEM         75
_A_VOLID          83
_MSC_VER          33+
__SC__            33
__TURBOC__                24
_chmod            41     158     165
_far              31
argc             178     182     184     202
args              46      50      51
argv             178     186     189     190     194     195     204     206     209     224
astr             156     167     175
atr               45      51      55      62     112     114     133     135
atr_clrmask               93     138     141     144     147     164
atr_setmask               92     117     120     123     126     163
attrib            91     158     163     164     165+    168     169     170     171     172     173
ax                53
buf              209     216     218     219     220
cflag             59
clrattr          133     195
ctype             19
cx                55      63
defined           24      33+
dos               18
ds                57
dx                54
exit             109     161
far               26
fclose           222
fgets            216
flag              94     165     191     196
fopen            211
fp                37+     38+    208     211     216     222
func              41      48+     50      53      61
h        16       17      18      19      20      25      28
i       180      184+    186     189     190     194     195     202+    204     206     209     224
intdosx           58
io                25
j       180      189+    190     194+    195
main             178
path              41      54      57     154     158     160     165     175
printf           160     175     213
ptr              156     167     168     169     170     171     172     173     174     209     211
         213
puts              98      99     100     101     102     103     104     105     106     107     108
regs              43      53      54      55      58+     59      63
s        22+
segread           56
setattr          112     190
show_atr         154     220     224
sregs             44      56      57      58
stdarg            28
stdio             16
stdlib            17
strchr           204
string            20
strlen            22
strupr           158
toupper          114     135
usage             96     129     150     183
va_arg            51
va_list           46
va_start          50
x        30+      53      54      55      59      63
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ========================================================================
  4 |     CLOCK.H     A simple macro based implementation of the ANSI/ISO
  5 |                 standard clock() function and related items.
  6 |
  7 |                 Some older C compilers for DOS do not implement these
  8 |                 items. E.g.: Turbo C 1.5 ... This file fixes it.
  9 |                 WARNING: There is no provision for handling the
 10 |                 24-hour/midnight rollover problem.
 11 |
 12 |                 Suggested use: #include this file _after_ time.h if
 13 |                 there are problems related to clock().
 14 |
 15 |                 v1.00  95-10-19  Initial version.
 16 |
 17 |  _____          This version is Public Domain.
 18 |  /_|__|         Auke Reitsma, Delft, The Netherlands.
 19 | /  | \  ------------------------------------------------------------ */
 20 |
 21 | #ifndef CLOCKS_PER_SEC
 22 |
 23 | #include "extkword.h"   /* from Snippets, for definition of FAR */
 24 |
 25 | typedef long clock_t;
 26 | #define clock()         *((clock_t FAR *)0x0040006cL)
 27 | #define CLOCKS_PER_SEC  18.2
 28 | #define CLK_TCK         18.2   /* Non-ANSI/ISO but _very_ common. */
 29 |
 30 | #endif
 31 |
 32 | /* === clock.h end ========================================================= */
```

TEXT STATISTICS

```
  1233 characters
    32 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    CLK_TCK         28
    CLOCKS_PER_SEC          21      27
    FAR             26
    clock           26
    clock_t         25      26
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  CMDLINE.C - Demonstrates accessing command line arguments
  5 | */
  6 |
  7 | #include <stdio.h>
  8 | #include <stdlib.h>
  9 | #include "snip_str.h"         /* For plural_text() and plural_text2()   */
 10 |
 11 | main(int argc, char *argv[])
 12 | {
 13 |       int i, n = argc - 1;
 14 |
 15 |       printf("You passed %d argument%s on the command line.",
 16 |             n, plural_text(n));
 17 |
 18 |       if (argc > 1)
 19 |       {
 20 |             puts(" They are:");
 21 |             for (i = 1; i < argc; ++i)
 22 |             {
 23 |                   printf("\nArgument #%d:\n  Text: \"%s\"\n  Value: %d\n",
 24 |                         i, argv[i], atoi(argv[i]));
 25 |             }
 26 |       }
 27 |       else  putchar('\n');
 28 |       return 0;
 29 | }
```

TEXT STATISTICS

```
   716 characters
    29 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     1 constants [character]
     4 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    argc            11      13      18      21
    argv            11      24+
    atoi            24
    h        7       8
    i       13      21+     24+
    main            11
    n       13      16+
    plural_text             16
    printf          15      23
    putchar         27
    puts            20
    stdio            7
    stdlib           8
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#) CMPSTR  - Comparing strings with variable sort order
* @(#) TCMPSTR - Comparing strings with variable sort order and
*                masking/truncation
*
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*1994-03-15/Bac
*   TCMPSTR added
*
*****************************************************************************
*1994/Erik Bachmann (E-mail: ebp@dde.dk)
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <string.h>                    /* strlen */

/**
#define TEST
**/

/*
 /-------------------------------------\
|          CMPSTR                      |-----------------------------------|
|\-------------------------------------/
|
| Comparing two strings with variable sort order
|
|
| Examples of sort sequences found in SORTKEY.h
|
|
|
|---------------------------------------------------------------------------|
| CALL:
|    iOrder = cmpstr("abc‘›†", "abc†›‘",
|                    (unsigned char *) CaseMatch) ;
|
| HEADER:
|    sortkey.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    char *pszStr1      Pointer to first string
|    char *pszStr2      Pointer to second string
|    char *pszOrder     Pointer to table with sort sequences
|
| PROTOTYPE:
|    int far cmpstr(unsigned char *pszStr1, unsigned char *pszStr2,
|                   unsigned char *pszOrder) ;
|
| RETURN VALUE:
|    -1          pszStr1 is first
|    0           the two strings are identical (i sort order)
|    1           pszStr2 is first
|
| MODULE:
|    cmpstr.c
|---------------------------------------------------------------------------|
|1993-06-07/Bac   Length comparision uptimised (length calculation excluded)

|
|---------------------------------------------------------------------------|
|1993-04-10/Erik Bachmann
\---------------------------------------------------------------------------|*/

int far cmpstr(unsigned char *pszStr1,
               unsigned char *pszStr2,
               unsigned char *pszOrder)
{
      int i = 0;

      /*---------------------------------------------------------*/

      while (('\0' != pszStr1[i]) && ('\0' != pszStr2[i]))
      {
            if (pszOrder[pszStr1[i]] < pszOrder[pszStr2[i]])
                  return -1;        /* Char from first string is first  */

            if (pszOrder[pszStr1[i]] > pszOrder[pszStr2[i]])
                  return 1;         /* Char from second string is first */

            i++;                    /* Next char                        */
      }

      if (('\0' == pszStr1[i] ) && ('\0' != pszStr2[i]))
            return -1;              /* First string is shortest         */

      if (('\0' != pszStr1[i] ) && ('\0' == pszStr2[i]))
            return 1;               /* Second string is shortest        */

      return 0;                     /* They are identical               */
}

/*
```

```
/*-------------------------------------\
|          TCMPSTR                     |-----------------------------------|
|\-------------------------------------/
|
| Comparing two strings with variable sort order and truncation/masking
|
|
| Examples of sort sequences found in SORTKEY.h
|
|
|
|-------------------------------------------------------------------------------|
| CALL:
|    iOrder = cmpstr("ab$‘›†?", "abc†›‘123",
|                    (unsigned char *) CaseMatch, "$", "?") ;
|
| HEADER:
|    sortkey.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    char *pszStr1      Pointer to first string
|    char *pszStr2      Pointer to second string
|    char *pszOrder     Pointer to table with sort sequences
|
| PROTOTYPE:
|    int far cmpstr(unsigned char *pszStr1,  unsigned char *pszStr2,
|                   unsigned char *pszOrder, unsigned char *pszMask,
|                   unsigned char *pszTrunc) ;
|
| RETURN VALUE:
|    -1          pszStr1 is first
|    0           the two strings are identical (i sort order)
|    1           pszStr2 is first
|
| MODULE:
|    cmpstr.c
|-------------------------------------------------------------------------------|
|
|-------------------------------------------------------------------------------|
|1994-03-14/Erik Bachmann
\-------------------------------------------------------------------------------|*/

int far tcmpstr(unsigned char *pszStr1,
                unsigned char *pszStr2,
                unsigned char *pszOrder,
                unsigned char *pszMask,
                unsigned char *pszTrunc)
{
      int  i = 0;
      char cTruncateFlag = 0;
      int  iCompareLength;

      /*-----------------------------------------------------------*/

      iCompareLength = strlen(pszStr1) - 1 ;
                                   /* Find last charater i string 1    */

      if (strrchr(pszTrunc, pszStr1[iCompareLength])
          && (0 <= iCompareLength))
                                   /* While string is terminated by one */
      {                            /* of the specified characters       */

            cTruncateFlag++;
            iCompareLength--;
      }

      while ((((0 != cTruncateFlag) && (i < iCompareLength)
               || (0 == cTruncateFlag) && ('\0' != pszStr1[i])))
             && ('\0' != pszStr2[i]))
      {
            if (0 != strchr(pszMask, pszStr1[i]))
            {
                  i++;
                  continue;        /* If string masked: continue       */
            }

            if (pszOrder[pszStr1[i]] < pszOrder[pszStr2[i]])
                  return -1;       /* Char from first string is first  */

            if (pszOrder[pszStr1[i]] > pszOrder[pszStr2[i]])
                  return 1;        /* Char from second string is first */

            i++;                   /* Next char                        */
      }

      if ((('\0' == pszStr1[i]) && ('\0' != pszStr2[i]))
          && (0 == cTruncateFlag))
      {
            return -1;             /* First string is shortest         */
      }

      if ((('\0' != pszStr1[i]) && ('\0' == pszStr2[i]))
          && 0 == cTruncateFlag)
      {
            return 1;              /* Second string is shortest        */
      }

|
```

```
201 |       return 0;                    /* They are identical              */
202 | }
203 |
204 |
205 | #ifdef TEST
206 |
207 | #include "sortkey.h"
208 |
209 | void state(int iOrder)
210 | {
211 |       if (0 == iOrder)
212 |             printf("\tequal");
213 |       else
214 |       {
215 |             if (0 <= iOrder)
216 |                   printf("\tString 1 is higher than string 2");
217 |             else  printf("\tString 1 is lower than string 2");
218 |       }
219 | }
220 |
221 | void main(void)
222 | {
223 |       int iOrder = 0;
224 |       char  *str1a   = "ab$‘›†?",
225 |             *str1b   = "ab$‘›†",
226 |             *str1c   = "abc‘›†?",
227 |             *str1d   = "ab$‘›†",
228 |             *str1e   = "abc†‘›",
229 |             *str1f   = "abc‘›†_",
230 |
231 |             *str2a   = "abc‘›†123",
232 |             *str2b   = "abc‘›†";
233 |
234 |       printf("\n\n");
235 |
236 |       /* 2a */
237 |
238 |       iOrder = tcmpstr(str1a, str2a, (unsigned char *) CaseMatch, "$", "?");
239 |       printf("\n%10s / %10s", str1a, str2a);
240 |       state(iOrder);
241 |
242 |       iOrder = tcmpstr(str1b, str2a, (unsigned char *) CaseMatch, "$", "?");
243 |       printf("\n%10s / %10s", str1b, str2a);
244 |       state(iOrder);
245 |
246 |       iOrder = tcmpstr(str1c, str2a, (unsigned char *) CaseMatch, "$", "?");
247 |       printf("\n%10s / %10s", str1c, str2a);
248 |       state(iOrder);
249 |
250 |       iOrder = tcmpstr(str1d, str2a, (unsigned char *) CaseMatch, "$", "?");
251 |       printf("\n%10s / %10s", str1d, str2a);
252 |       state(iOrder);
253 |
254 |       iOrder = tcmpstr(str1e, str2a, (unsigned char *) CaseMatch, "$", "?");
255 |       printf("\n%10s / %10s", str1e, str2a);
256 |       state(iOrder);
257 |
258 |       iOrder = tcmpstr(str1f, str2a, (unsigned char *) CaseMatch, "$", "?");
259 |       printf("\n%10s / %10s", str1f, str2a);
260 |       state(iOrder);
261 |
262 |       /* 2b */
263 |
264 |       iOrder = tcmpstr(str1a, str2b, (unsigned char *) CaseMatch, "$", "?");
265 |       printf("\n%10s / %10s", str1a, str2b);
266 |       state(iOrder);
267 |
268 |       iOrder = tcmpstr(str1b, str2b, (unsigned char *) CaseMatch, "$", "?");
269 |       printf("\n%10s / %10s", str1b, str2b);
270 |       state(iOrder);
271 |
272 |       iOrder = tcmpstr(str1c, str2b, (unsigned char *) CaseMatch, "$", "?");
273 |       printf("\n%10s / %10s", str1c, str2b);
274 |       state(iOrder);
275 |
276 |       iOrder = tcmpstr(str1d, str2b, (unsigned char *) CaseMatch, "$", "?");
277 |       printf("\n%10s / %10s", str1d, str2b);
278 |       state(iOrder);
279 | }
280 | #endif
```

TEXT STATISTICS

```
  8714 characters
   280 lines
```

LEXICAL STATISTICS

```
    26 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
    12 constants [character]
    43 constants [string]
    23 constants [numeric]
```

SYMBOL TABLE

```
    CaseMatch       238     242     246     250     254     258     264     268     272     276
    TEST            205
    cTruncateFlag           153     166     170     171     190     196
    cmpstr           72
    far              72     146
    h        19
    i        76      80+     82+     85+     88      91+     94+     152     170     171     172     174
            176     180+    183+    186     189+    195+
    iCompareLength          154     158     161     162     167     170
    iOrder          209     211     215     223     238     240     242     244     246     248     250
            252     254     256     258     260     264     266     268     270     272     274     276
            278
    main            221
    printf          212     216     217     234     239     243     247     251     255     259     265
            269     273     277
    pszMask         149     174
    pszOrder         74      82+     85+     148     180+    183+
    pszStr1          72      80      82      85      91      94      146     158     161     171     174
            180     183     189     195
    pszStr2          73      80      82      85      91      94      147     172     180     183     189
            195
    pszTrunc        150     161
    state           209     240     244     248     252     256     260     266     270     274     278
    str1a           224     238     239     264     265
    str1b           225     242     243     268     269
    str1c           226     246     247     272     273
    str1d           227     250     251     276     277
    str1e           228     254     255
    str1f           229     258     259
    str2a           231     238     239     242     243     246     247     250     251     254     255
            258     259
    str2b           232     264     265     268     269     272     273     276     277
    strchr          174
    string           19
    strlen          158
    strrchr         161
    tcmpstr         146     238     242     246     250     254     258     264     268     272     276
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 |  * CMTCONVR.H
  5 |  * Comment converter class.
  6 |  *
  7 |  * ver 1.0, 30 Jun 1996
  8 |  *
  9 |  * Public domain by:
 10 |  *   Jari Laaksonen
 11 |  *   Arkkitehdinkatu 30 A 2
 12 |  *   FIN-33720 Tampere
 13 |  *   FINLAND
 14 |  *
 15 |  *   Fidonet : 2:221/360.20
 16 |  *   Internet: jla@to.icl.fi
 17 |  */
 18 |
 19 | #ifndef _CMTCONVR_H_
 20 | #define _CMTCONVR_H_
 21 |
 22 | #include "cmtparsr.h"
 23 |
 24 | class CommentConverter : public CommentParser
 25 | {
 26 | public:
 27 |
 28 | protected:
 29 |     virtual void ProcessActions (Event theEvent);
 30 |
 31 | };
 32 |
 33 | #endif // _CMTCONVR_H_
 34 |
```

TEXT STATISTICS

```
   534 characters
    34 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     1 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CommentConverter        24
    CommentParser           24
    Event           29
    ProcessActions          29
    _CMTCONVR_H_            19      20
    theEvent        29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * CMTCOUNT.H
 * Comment counter class.
 *
 * ver 1.0, 30 Jun 1996
 *
 * Public domain by:
 *   Jari Laaksonen
 *   Arkkitehdinkatu 30 A 2
 *   FIN-33720 Tampere
 *   FINLAND
 *
 *   Fidonet : 2:221/360.20
 *   Internet: jla@to.icl.fi
 */

#ifndef _CMTCOUNT_H_
#define _CMTCOUNT_H_

#include "cmtparsr.h"

class CommentCounter : public CommentParser
{
public:
    CommentCounter()
        : CommentParser()
        , cpp_comments(0)
        , open_comment(0)
        , close_comment(0)
        , cmt_lines(0)
    { }

    int GetCppComments()  { return cpp_comments; }
    int GetOpenComment()  { return open_comment; }
    int GetCloseComment() { return close_comment; }
    int GetCommentLines() { return cmt_lines; }

protected:
    virtual void ProcessActions (Event theEvent);

private:
    int cpp_comments;
    int cmt_lines;
    int open_comment,
        close_comment;
};

#endif // _CMTCOUNT_H_
```

```
TEXT STATISTICS

  1002 characters
    51 lines

LEXICAL STATISTICS

     2 comments [std-C]
     1 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    CommentCounter          24      27
    CommentParser           24      28
    Event           41
    GetCloseComment         37
    GetCommentLines         38
    GetCppComments          35
    GetOpenComment          36
    ProcessActions          41
    _CMTCOUNT_H_            19      20
    close_comment           31      37      47
    cmt_lines       32      38      45
    cpp_comments            29      35      44
    open_comment            30      36      46
    theEvent        41
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * CMTPARSR.H
 * Base class: comment parser.
 *
 * ver 1.0, 30 Jun 1996
 *
 * Public domain by:
 *   Jari Laaksonen
 *   Arkkitehdinkatu 30 A 2
 *   FIN-33720 Tampere
 *   FINLAND
 *
 *   Fidonet : 2:221/360.20
 *   Internet: jla@to.icl.fi
 */

#ifndef _CMTPARSR_H_
#define _CMTPARSR_H_

#include <stdio.h>

class CommentParser
{
public:
    enum State {
        NormalInput,
        InsideString,
        InsideChar,
        BeginComment,
        InsideEscape,
        InCppComment,
        InCComment,
        StarInCppComment,
        StarInCComment
    };

    enum Event {
        ANY_CHAR,
        FOUND_BACKSLASH,
        FOUND_NL,
        FOUND_QUOTE,
        FOUND_SINGLEQUOTE,
        FOUND_SLASH,
        FOUND_STAR,
        FOUND_WHITESPACE,
        END_OF_FILE
    };

    CommentParser (State InitState = NormalInput)
        : itsState(InitState)
        , lines(1)
    { }

    virtual ~CommentParser() { }

    int Init (int argc, char **argv);
    int Init (char *file);

    void Uninit();

    int Run();

    unsigned long GetLines() { return lines - 1; };

    int GetLastState() { return itsState; };

protected:
    virtual void ProcessState (Event theEvent);

    virtual void ProcessActions (Event theEvent) = 0;

    void print (char ch)
    {
        fputc (ch, OutFile);
    }

    void print (char *str)
    {
        fputs (str, OutFile);
    }

    void ChangeChar (int ch)
    {
        ReadCh = ch;
    }

    int ReadChar();
    void PrintChar();
    void PrintLineNumber();

    Event GetEvent();

    State   itsState;
    State   itsPrevState;

private:
    unsigned long  lines;
    int     arguments;
```

```
 101 |     int     ReadCh;
 102 |     FILE   *InFile,
 103 |            *OutFile;
 104 | };
 105 |
 106 | #endif // _CMTPARSR_H_
 107 |
```

TEXT STATISTICS

```
  1882 characters
   107 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     1 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    ANY_CHAR        40
    BeginComment            31
    ChangeChar              84
    CommentParser           24      51      56
    END_OF_FILE             48
    Event           39      70      72      93
    FILE           102
    FOUND_BACKSLASH         41
    FOUND_NL        42
    FOUND_QUOTE             43
    FOUND_SINGLEQUOTE       44
    FOUND_SLASH             45
    FOUND_STAR              46
    FOUND_WHITESPACE        47
    GetEvent        93
    GetLastState            67
    GetLines        65
    InCComment              34
    InCppComment            33
    InFile         102
    Init            58      59
    InitState       51      52
    InsideChar              30
    InsideEscape            32
    InsideString            29
    NormalInput             28      51
    OutFile         76      81     103
    PrintChar       90
    PrintLineNumber         91
    ProcessActions          72
    ProcessState            70
    ReadCh          86     101
    ReadChar        89
    Run             63
    StarInCComment          36
    StarInCppComment        35
    State           27      51      95      96
    Uninit          61
    _CMTPARSR_H_            19      20
    argc            58
    arguments      100
    argv            58
    ch              74      76      84      86
    file            59
    fputc           76
    fputs           81
    h       22
    itsPrevState            96
    itsState        52      67      95
    lines           53      65      99
    print           74      79
    stdio           22
    str             79      81
    theEvent        70      72
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 |  * CMTREMOV.H
  5 |  * Comment remover class.
  6 |  *
  7 |  * ver 1.0, 30 Jun 1996
  8 |  *
  9 |  * Public domain by:
 10 |  *   Jari Laaksonen
 11 |  *   Arkkitehdinkatu 30 A 2
 12 |  *   FIN-33720 Tampere
 13 |  *   FINLAND
 14 |  *
 15 |  *   Fidonet : 2:221/360.20
 16 |  *   Internet: jla@to.icl.fi
 17 |  */
 18 |
 19 | #ifndef _CMTREMOV_H_
 20 | #define _CMTREMOV_H_
 21 |
 22 | #include "cmtparsr.h"
 23 |
 24 | class CommentRemover : public CommentParser
 25 | {
 26 | public:
 27 |
 28 | protected:
 29 |     virtual void ProcessActions (Event theEvent);
 30 |
 31 | };
 32 |
 33 | #endif // _CMTREMOV_H_
 34 |
```

TEXT STATISTICS

```
   530 characters
    34 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     1 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CommentParser           24
    CommentRemover          24
    Event           29
    ProcessActions          29
    _CMTREMOV_H_            19      20
    theEvent        29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * Test file for Comment Utilities.
 *   This file should be compilable before AND after comment
 *   conversion or removal.
 *
 *   Jari Laaksonen
 *   Arkkitehdinkatu 30 A 2
 *   FIN-33720 Tampere
 *   FINLAND
 *   Fidonet:  2:221/360.20
 *   Internet: jla@to.icl.fi
 */

#include <stdio.h>

int main()
{
/*
  comment */ int a; /* comment continues in same line
*/

/* C comment */ int b; // C++ comment

/*
   C comment
*/ int c; // C++ comment

  /**** comment ****/
  //*** comment ****/

  char ch = '\"'; /* double quote but not a start of a string */

  char x1[]       // here we...
    = "";
  char x2[] = ""; /* ...have some... */
  char x3[] = ""  /* ...empty... */ ;
  char x4[] = ""; // ...strings.

  printf ("this is a string");          /* C comment */
  printf ("this is \" another string"); // C++ comment
  printf ("this is \' another string"); // C++ comment
  printf ("yet another \\ string");     // C++ comment

/* C comment in one line */

// C++ comment in one line

/* C comment
   in several
   lines
   printf ("// not a comment");
*/

/* C comment
   in several lines */

// C comment in C++ comment: /* comment */

/* C++ comment in C comment: // comment */

/*
   C++ comment in C comment: // comment
*/

  printf ("this /* is not // a comment * * ! ");
  printf ("this /* is not a comment * * ! ");
  printf ("this // is not a comment * * ! ");

  printf ("this */ is not a comment * * ! ");

// C++ comment
// C++ comment /

// C++ comment in \
   several        \
   lines

/\
/ C++ comment

  a = 0; /\
* C comment */

  /* C       *\
   *         *
  \* comment */

/* C comment \
   C comment */

// char s[] = "string \
               string";

// not a multiline C++ \comment
  b = 0;

// not a multiline C++ \ comment
  c = 0;
```

```
 101 |
 102 |   return 0;
 103 | }
 104 | // end file


TEXT STATISTICS

  1944 characters
   104 lines

LEXICAL STATISTICS

    18 comments [std-C]
    17 comments [C++]
     1 preprocessor instructions
     1 constants [character]
    12 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    C       81      84
    a       21      83
    b       24      97
    c       28     100
    ch              33
    comment         81      84
    h       16
    lines           78
    main            18
    printf          41      42      43      44      67      68      69      71
    several         77
    stdio           16
    string"         94
    x1              35
    x2              37
    x3              38
    x4              39
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 |  * CMTXTRAC.H
   5 |  * Comment extractor class.
   6 |  *
   7 |  * ver 1.0, 30 Jun 1996
   8 |  *
   9 |  * Public domain by:
  10 |  *   Jari Laaksonen
  11 |  *   Arkkitehdinkatu 30 A 2
  12 |  *   FIN-33720 Tampere
  13 |  *   FINLAND
  14 |  *
  15 |  *   Fidonet : 2:221/360.20
  16 |  *   Internet: jla@to.icl.fi
  17 |  */
  18 |
  19 | #ifndef _CMTXTRAC_H_
  20 | #define _CMTXTRAC_H_
  21 |
  22 | #include "cmtparsr.h"
  23 |
  24 | class CommentExtractor : public CommentParser
  25 | {
  26 | public:
  27 |     CommentExtractor()
  28 |         : CommentParser()
  29 |         , printWhiteSpace(0)
  30 |         , printLineNumbers(0)
  31 |     { }
  32 |
  33 |     void WhiteSpace  (int OnOff) { printWhiteSpace  = OnOff; }
  34 |     void LineNumbers (int OnOff) { printLineNumbers = OnOff; }
  35 |
  36 | protected:
  37 |     virtual void ProcessActions (Event theEvent);
  38 |
  39 | private:
  40 |     int printWhiteSpace;
  41 |     int printLineNumbers;
  42 | };
  43 |
  44 | #endif // _CMTXTRAC_H_
  45 |
```

TEXT STATISTICS

```
   848 characters
    45 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     1 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    CommentExtractor        24      27
    CommentParser           24      28
    Event           37
    LineNumbers             34
    OnOff           33+     34+
    ProcessActions          37
    WhiteSpace              33
    _CMTXTRAC_H_            19      20
    printLineNumbers        30      34      41
    printWhiteSpace         29      33      40
    theEvent        37
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Compute C(n,m) = the number of combinations of n items,
** taken m at a time.
**
** Written by Thad Smith III, Boulder County, CO.
** Released to the Public Domain  10/14/91.
**
** The def of this function is
**      C(n,m)  = n! / (m! * (n-m)!).
** Computing this formula can cause overflow for large values of n,
** even when C(n,m) would be defined.
**
** The first version will not overflow if C(n,m) * (n-m+1) < ULONG_MAX.
** The second version will not overflow if C(n,m) < ULONG_MAX, but
** is slightly more complex.
** Function domain: n >= 0,  0 <= m <= n.
**
** Both versions work by reducing the product as it is computed.  It
** relies on the property that the product on n consecutive integers
** must be evenly divisible by n.
**
** The first version can be changed to make cnm and the return value
** double to extend the range of the function.
*/

#include "snipmath.h"

DWORD ncomb1 (int n, int m)
{
      DWORD cnm = 1UL;
      int i;

      if (m*2 >n) m = n-m;
      for (i=1 ; i <= m; n--, i++)
            cnm = cnm * n / i;
      return cnm;
}

DWORD ncomb2 (int n, int m)
{
      DWORD cnm = 1UL;
      int i, f;

      if (m*2 >n) m = n-m;
      for (i=1 ; i <= m; n--, i++)
      {
            if ((f=n) % i == 0)
                  f   /= i;
            else  cnm /= i;
            cnm *= f;
      }
      return cnm;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif
#ifdef __TURBOC__
 #pragma argsused
#endif

main (int argc, char *argv[]) {
    int n,m;
    n = atoi (argv[1]);
    m = atoi (argv[2]);
    printf ("ncomb1 = %lu, ncomb2 = %lu\n", ncomb1(n,m), ncomb2(n,m));
    return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1823 characters
    77 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     2 constants [string]
    10 constants [numeric]

SYMBOL TABLE

    DWORD           30      32      41      43
    TEST            57
    __TURBOC__              65
    __WATCOMC__             62
    argc            69
    argsused        66
    argv            69      71      72
    atoi            71      72
    cnm             32      37+     38      43      51      52      54
    f       44      49      50      52
    h       59      60
    i       33      36+     37      44      47+     49      50      51
    m       30      35+     36      41      46+     47      70      72      73+
    main            69
    n       30      35+     36      37      41      46+     47      49      70      71      73+
    ncomb1          30      73
    ncomb2          41      73
    off             63
    printf          73
    stdio           59
    stdlib          60
    unreferenced            63
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  COMMAFLT.C - Format a double using commas as thousands separators
**               and a specified number of significant fractional digits.
**
**  public domain by Bruce Wedding and Kurt Kuzba
*/
#include <stdio.h>

#include "numcnvrt.h"
#include "snip_str.h"

char *comma_float( double num, char *buf, int dec)
{
      char tmp[80];
      int bf = 0, cm = 0, tm = 9 - dec + (!dec);

      sprintf(tmp, "%.9f", num);
      strrev(tmp);
      if(dec)
      {
            while( (buf[bf++] = tmp[tm++]) != '.')
                  ;
            while((buf[bf++] = tmp[tm++]) != 0)
            {
                  if(++cm % 3 == 0 && tmp[tm])
                        buf[bf++] = ',';
            }
      }
      return strrev(buf);
}

#ifdef TEST

#include <stdio.h>

int main( void )
{
      int i;
      char result[80];
      double num[10] = {
            1.98765,       12.98765,       123.98765,
            1234.98765,    12345.98765,    123456.98765,
            1234567.98765, 12345678.98765, 123456789.98765,
            1234567890.98765  };

      for ( i = 0; i < 10; i++ )
            printf("Before: %-24.5f After: %s \n",
                  num[i], comma_float(num[i], result, 6));
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1331 characters
    54 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     2 constants [character]
     4 constants [string]
    23 constants [numeric]

SYMBOL TABLE

    TEST            34
    bf              17      23      25      28
    buf             14      23      25      28      31
    cm              17      27
    comma_float             14      50
    dec             14      17+     21
    h        9      36
    i       40      48+     50+
    main            38
    num             14      19      42      50+
    printf          49
    result          41      50
    sprintf         19
    stdio            9      36
    strrev          20      31
    tm              17      23      25      27
    tmp             16      19      20      23      25      27
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  COMMAFMT.C
**
**  Public domain by Bob Stout, bug fixes by Mark Kamradt
**
**  Notes:  1. Use static buffer to eliminate error checks on buffer overflow
**             and reduce code size.
**          2. By making the numeric argument a long and prototyping it before
**             use, passed numeric arguments will be implicitly cast to longs
**             thereby avoiding int overflow.
**          3. Use the thousands grouping and thousands separator from the
**             ANSI locale to make this more robust.
*/

#include <string.h>
#include "numcnvrt.h"

size_t commafmt(char   *buf,            /* Buffer for formatted string  */
                int     bufsize,        /* Size of buffer               */
                long    N)              /* Number to convert            */
{
      int len = 1, posn = 1, sign = 1;
      char *ptr = buf + bufsize - 1;

      if (2 > bufsize)
      {
ABORT:      *buf = NUL;
            return 0;
      }

      *ptr-- = NUL;
      --bufsize;
      if (0L > N)
      {
            sign = -1;
            N = -N;
      }

      for ( ; len <= bufsize; ++len, ++posn)
      {
            *ptr-- = (char)((N % 10L) + '0');
            if (0L == (N /= 10L))
                  break;
            if (0 == (posn % 3))
            {
                  *ptr-- = ',';
                  ++len;
            }
            if (len >= bufsize)
                  goto ABORT;
      }

      if (0 > sign)
      {
            if (len >= bufsize)
                  goto ABORT;
            *ptr-- = '-';
            ++len;
      }

      memmove(buf, ++ptr, len + 1);
      return (size_t)len;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif
#ifdef __TURBOC__
 #pragma argsused
#endif

main(int argc, char *argv[])
{
      size_t len;
      char buf[20];
      long N;

      N = strtol(argv[1], NULL, 10);
      len = commafmt(buf, 20, N);
      printf("%s converts to %s and returned %d\n", argv[1], buf, len);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2196 characters
    91 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     3 constants [character]
     2 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    ABORT           29      52      58
    EXIT_SUCCESS            88
    N       22      35      38+     43      44      83      85      86
    NUL             29      33
    NULL            85
    TEST            67
    __TURBOC__              75
    __WATCOMC__             72
    argc            79
    argsused        76
    argv            79      85      87
    buf             20      25      29      63      82      86      87
    bufsize         21      25      27      34      41      51      57
    commafmt        20      86
    h       17      69      70
    len             24      41+     49      51      57      60      63      64      81      86      87
    main            79
    memmove         63
    off             73
    posn            24      41      46
    printf          87
    ptr             25      33      43      48      59      63
    sign            24      37      55
    size_t          20      64      81
    stdio           69
    stdlib          70
    string          17
    strtol          85
    unreferenced            73
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * COMMCONV.C
 * Change C++ comments to C comments
 *
 * ver 2.2, 14 Jun 1995
 *   - changed multiline C++ comment handling.
 *
 * ver 2.3, 8 Nov 1995
 *   - bug: if file was ended with // comment with no CR/LF,
 *          program jammed in an endless loop.
 *
 * ver 2.4, 21 Nov 1995
 *   - bug: did not handle escape character \" inside strings.
 *
 * Public domain by:
 *   Jari Laaksonen
 *   Arkkitehdinkatu 30 A 2
 *   FIN-33720 Tampere
 *   FINLAND
 *
 *   Fidonet : 2:221/360.20
 *   Internet: jla@to.icl.fi
 */

#include <stdio.h>

int main (int argc, char **argv)
{
  int  Char,
       Char2,
       got_endl      = 0,
       cpp_comment   = 0,
       c_comment     = 0,
       in_string     = 0,
       cpp_multiline = 0;
  char CannotOpen[] = "Cannot open %s\n\n";
  FILE *InFile, *OutFile = stdout;

  if (argc < 2)
  {
    fprintf (stderr, "USAGE: COMMCONV InFile [OutFile]\n");
    return 1;
  }
  if ((InFile = fopen (argv[1], "r")) == NULL)
  {
    fprintf (stderr, CannotOpen, argv[1]);
    return 2;
  }

  if (argc == 3)
  {
    if ((OutFile = fopen (argv[2], "w")) == NULL)
    {
      fprintf (stderr, CannotOpen, argv[2]);
      OutFile = stdout;  /* if can't open, output goes to stdout instead */
    }
  }

  while ((Char = fgetc (InFile)) != EOF)
  {
    fputc (Char, OutFile);

    if (in_string && Char == '\\')
    {
      Char2 = fgetc (InFile);
      fputc (Char2, OutFile);

      /* do we have an excape character \" inside string ? */
      if (Char2 == '\"')
        continue;
    }

    if (Char == '\"')
    {
      Char2 = fgetc (InFile);            /* check next char */

      /* do we have a character constant '\"' or an empty string "" ? */
      if (Char2 != '\'' && Char2 != '\"')
        in_string = ! in_string;         /* if not, we are in a string */
      fputc (Char2, OutFile);
    }

    if (in_string)                       /* we are in a string now */
      continue;

    if (Char == '/')
    {
      Char = fgetc (InFile);             /* check next char */
      if (Char == '/')                   /* is it start of C++ comment? */
      {
        Char = '*';                      /* change it to C comment */
        cpp_comment = 1;
      }
      else if (Char == '*')              /* is it start of C comment? */
        c_comment = 1;

      fputc (Char, OutFile);

```

```
101 |     }
102 |     else if (Char == '*' && c_comment)
103 |     {
104 |       Char = fgetc (InFile);
105 |       if (Char == '/')                  /* is it end of C comment? */
106 |         c_comment = 0;
107 |       fputc (Char, OutFile);
108 |     }
109 |
110 |     if (c_comment || cpp_comment)       /* are we inside C or C++ comment? */
111 |     {
112 |       got_endl = 1;
113 |       while ((Char = fgetc (InFile)) != '\n') /* check the rest of the line */
114 |       {
115 |         if (Char == EOF)
116 |           break;
117 |
118 |         if (cpp_multiline)
119 |         {
120 |           if (Char != ' ' && Char != '\t') /* if not white space => */
121 |             cpp_multiline = 0;            /* ...not C++ multiline comment */
122 |         }
123 |
124 |         if (Char == '\\' && cpp_comment)
125 |           cpp_multiline = 1;
126 |
127 |         if (Char == '*')
128 |         {
129 |           Char2 = fgetc (InFile);       /* check next char */
130 |           ungetc (Char2, InFile);       /* put it back to stream */
131 |
132 |           if (Char2 == '/')             /* is it end of C comment */
133 |           {
134 |             c_comment = 0;
135 |             /* is it end of C comment inside C++ comment */
136 |             if (cpp_comment)
137 |             {
138 |               fputs ("* ", OutFile);
139 |               Char = fgetc (InFile);
140 |             }
141 |           }
142 |         }
143 |         fputc (Char, OutFile);
144 |         if (c_comment == 0 && cpp_comment == 0)
145 |         {
146 |           got_endl = 0;
147 |           break;          /* break if end of C comment found */
148 |         }
149 |       } /* while */
150 |
151 |       if (cpp_comment && cpp_multiline == 0)
152 |       {
153 |         fputs (" */", OutFile);          /* put ending C comment mark */
154 |         cpp_comment = 0;
155 |       }
156 |
157 |       /* print endl if we processed whole line */
158 |       if (got_endl)
159 |         fputc ('\n', OutFile);
160 |
161 |       /* clear flag for the next round. if it is still clear after
162 |          next C++ comment line is processed, multiline C++ comment
163 |          is ended.
164 |       */
165 |       cpp_multiline = 0;
166 |     }
167 |   } /* while end */
168 |
169 |   if (argc == 3)
170 |     fclose (OutFile);
171 |   fclose (InFile);
172 |
173 |   fflush (stdout);
174 |   fprintf (stderr, "\nOK!\n");
175 |
176 |   return 0;
177 | }
```

```
TEXT STATISTICS

  4562 characters
   177 lines

LEXICAL STATISTICS

    27 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
    18 constants [character]
     7 constants [string]
    28 constants [numeric]

SYMBOL TABLE

    CannotOpen              38      48      56
    Char             31     61      63      65      75      88      90      91      93      96      99
           102      104     105     107     113     115     120+    124     127     139     143
    Char2            32     67      68      71      77      80+     82      129     130     132
    EOF              61     115
    FILE             39
    InFile           39     46      61      67      77      90      104     113     129     130     139
           171
    NULL             46     54
    OutFile          39     54      57      63      68      82      99      107     138     143     153
           159      170
    argc             29     41      52      169
    argv             29     46      48      54      56
    c_comment        35     97      102     106     110     134     144
    cpp_comment              34      94      110     124     136     144     151     154
    cpp_multiline            37      118     121     125     151     165
    fclose           170    171
    fflush           173
    fgetc            61     67      77      90      104     113     129     139
    fopen            46     54
    fprintf          43     48      56      174
    fputc            63     68      82      99      107     143     159
    fputs            138    153
    got_endl         33     112     146     158
    h        27
    in_string        36     65      81+     85
    main             29
    stderr           43     48      56      174
    stdio            27
    stdout           39     57      173
    ungetc           130
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Tests for popular PC compilers and versions
*/

#include <stdio.h>

main()
{
      int version;

#if defined(__ZTC__)
 #if defined(__SC__)
      printf("Symantec C++ ver. %x.%x\n", __SC__ >> 8, __SC__ & 0xff);
 #else
      printf("Zortech C++ ver. %x.%xr%x\n",
            __ZTC__ >> 8, (__ZTC__ >> 4) & 0xf, __ZTC__ & 0xf);
 #endif
#elif defined(__WATCOMC__)
      printf("Watcom C/C++ ver. %d.%d\n",
            __WATCOMC__ / 100, __WATCOMC__ % 100);
#elif defined(__TURBOC__)
      version = __TURBOC__;
      if (0x295 > version)
      {
            printf("Borland Turbo C ver. %x.%02x\n",
                  version >> 8, version & 0xff);
      }
      else if (0x400 <= version)
      {
 #if defined(__BORLANDC__)
            printf("Borland C++ ver. %x.%x\n",
                  (version >> 8) - 1, (version & 0xff) >> 4);
 #else
            printf("Borland Turbo C++ ver. %x.%x\n",
                  (version >> 8) - 1, (version & 0xff) >> 4);
 #endif
      }
      else if (0x297 > version)
            printf("Borland Turbo C++ ver. 1.%02x\n", version - 0x295);
      else  printf("Borland C++ ver. 2.%02x\n", version - 0x297);
#elif defined(_MSC_VER)
      printf("Microsoft C(/C++) ver. %d.%d\n",
            _MSC_VER / 100, _MSC_VER % 100);
#elif defined(_QC)
      printf("Microsoft Quick C ver. %d.%d\n", _QC / 100, _QC % 100);
#elif defined(__POWERC)
      printf ("MIX Power C ver. %d.%d.%d\n",
            __POWERC/100, (__POWERC / 10) % 10, __POWERC % 10);
#elif defined(__GNUC__)
 #ifdef __EMX__
      printf("GNU/EMX C version %d.%d\n", __GNUC__, __GNUC_MINOR__);
  #else
      printf("GNU C version %d.%d\n", __GNUC__, __GNUC_MINOR__);
 #endif
#elif defined(__IBMC__) || defined (__IBMCPP__)
 #ifdef __IBMC__
      version = __IBMC__;
 #else
      version = __IBMCPP__;
 #endif
      if (version >= 300)
            printf ("IBM VisualAge C++ %d.%d\n",
                  version / 100, version % 100);
      else  printf ("IBM C Set++ %d.%d\n", version / 100, version % 100);
#else
      puts("Unknown compiler!");
#endif
      return 0;
}
```

TEXT STATISTICS

```
  2181 characters
    71 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
    23 preprocessor instructions
     0 constants [character]
    16 constants [string]
    37 constants [numeric]
```

SYMBOL TABLE

```
    _MSC_VER        43      45+
    _QC             46      47+
    __BORLANDC__            32
    __EMX__         52
    __GNUC_MINOR__          53      55
    __GNUC__        51      53      55
    __IBMCPP__              57      61
    __IBMC__        57      58      59
    __POWERC        48      50+
    __SC__          14      15+
    __TURBOC__              23      24
    __WATCOMC__             20      22+
    __ZTC__         13      18+
    defined         13      14      20      23      32      43      46      48      51      57+
    h       7
    main             9
    printf          15      17      21      27      33      36      41      42      44      47      49
            53      55      64      66
    puts            68
    stdio            7
    version         11      24      25      28+     30      34+     37+     40      41      42      59
            61      63      65+     66+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  CRC.H - header file for SNIPPETS CRC and checksum functions
*/

#ifndef CRC__H
#define CRC__H

#include <stdlib.h>           /* For size_t                 */
#include "sniptype.h"         /* For BYTE, WORD, DWORD      */

/*
**  File: ARCCRC16.C
*/

void init_crc_table(void);
WORD crc_calc(WORD crc, char *buf, unsigned nbytes);
void do_file(char *fn);

/*
**  File: CRC-16.C
*/

WORD crc16(char *data_p, WORD length);

/*
**  File: CRC-16F.C
*/

WORD updcrc(WORD icrc, BYTE *icp, size_t icnt);

/*
**  File: CRC_32.C
*/

#define UPDC32(octet,crc) (crc_32_tab[((crc)\
     ^ ((BYTE)octet)) & 0xff] ^ ((crc) >> 8))

DWORD updateCRC32(unsigned char ch, DWORD crc);
Boolean_T crc32file(char *name, DWORD *crc, long *charcnt);
DWORD crc32buf(char *buf, size_t len);

/*
**  File: CHECKSUM.C
*/

unsigned checksum(void *buffer, size_t len, unsigned int seed);

/*
**  File: CHECKEXE.C
*/

void checkexe(char *fname);


#endif /* CRC__H */
```

TEXT STATISTICS

1043 characters
58 lines

LEXICAL STATISTICS

11 comments [std-C]
0 comments [C++]
6 preprocessor instructions
0 constants [character]
1 constants [string]
2 constants [numeric]

SYMBOL TABLE

BYTE	31	38			
Boolean_T	41				
CRC__H	7	8			
DWORD	40+	41	42		
UPDC32	37				
WORD	18+	25+	31+		
buf	18	42			
buffer	48				
ch	40				
charcnt	41				
checkexe	54				
checksum	48				
crc	18	37+	38	40	41
crc16	25				
crc32buf	42				
crc32file	41				
crc_32_tab		37			
crc_calc	18				
data_p	25				
do_file	19				
fn	19				
fname	54				
h 10					
icnt	31				
icp	31				
icrc	31				
init_crc_table		17			
len	42	48			
length	25				
name	41				
nbytes	18				
octet	37	38			
seed	48				
size_t	31	42	48		
stdlib	10				
updateCRC32		40			
updcrc	31				

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #define POLY 0x8408
  4 | /*
  5 | //                                      16   12   5
  6 | // this is the CCITT CRC 16 polynomial X  + X  + X  + 1.
  7 | // This works out to be 0x1021, but the way the algorithm works
  8 | // lets us use 0x8408 (the reverse of the bit pattern).  The high
  9 | // bit is always assumed to be set, thus we only use 16 bits to
 10 | // represent the 17 bit value.
 11 | */
 12 |
 13 | #include "crc.h"
 14 |
 15 | WORD crc16(char *data_p, WORD length)
 16 | {
 17 |       unsigned char i;
 18 |       unsigned int data;
 19 |       unsigned int crc = 0xffff;
 20 |
 21 |       if (length == 0)
 22 |             return (~crc);
 23 |
 24 |       do
 25 |       {
 26 |             for (i=0, data=(unsigned int)0xff & *data_p++;
 27 |                  i < 8;
 28 |                  i++, data >>= 1)
 29 |             {
 30 |                   if ((crc & 0x0001) ^ (data & 0x0001))
 31 |                         crc = (crc >> 1) ^ POLY;
 32 |                   else  crc >>= 1;
 33 |             }
 34 |       } while (--length);
 35 |
 36 |       crc = ~crc;
 37 |       data = crc;
 38 |       crc = (crc << 8) | ((data >> 8) & 0xff);
 39 |
 40 |       return (crc);
 41 | }
```

TEXT STATISTICS

```
  1071 characters
    41 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

```
    POLY             3      31
    WORD            15+
    crc             19      22      30      31+     32      36+     37      38+     40
    crc16           15
    data            18      26      28      30      37      38
    data_p          15      26
    i       17      26      27      28
    length          15      21      34
```

```
/* +++Date last modified: 05-Jul-1997 */


/*
 * Calculate, intelligently, the CRC of a dataset incrementally given a
 * buffer full at a time.
 * Initialize crc to 0 for XMODEM, -1 for CCITT.
 *
 * Usage:
 *   newcrc = updcrc( oldcrc, bufadr, buflen )
 *        size_t oldcrc, buflen;
 *        char *bufadr;
 *
 * Compile with -DTEST to generate program that prints CRC of stdin to stdout.
 * Compile with -DMAKETAB to print values for crctab to stdout
 */

#include "crc.h"

    /* the CRC polynomial. This is used by XMODEM (almost CCITT).
     * If you change P, you must change crctab[]'s initial value to what is
     * printed by initcrctab()
     */
#define   P    0x1021

    /* number of bits in CRC: don't change it. */
#define W 16

    /* this the number of bits per char: don't change it. */
#define B 8

static WORD crctab[1<<B] = { /* as calculated by initcrctab() */
    0x0000,  0x1021,  0x2042,  0x3063,  0x4084,  0x50a5,  0x60c6,  0x70e7,
    0x8108,  0x9129,  0xa14a,  0xb16b,  0xc18c,  0xd1ad,  0xe1ce,  0xf1ef,
    0x1231,  0x0210,  0x3273,  0x2252,  0x52b5,  0x4294,  0x72f7,  0x62d6,
    0x9339,  0x8318,  0xb37b,  0xa35a,  0xd3bd,  0xc39c,  0xf3ff,  0xe3de,
    0x2462,  0x3443,  0x0420,  0x1401,  0x64e6,  0x74c7,  0x44a4,  0x5485,
    0xa56a,  0xb54b,  0x8528,  0x9509,  0xe5ee,  0xf5cf,  0xc5ac,  0xd58d,
    0x3653,  0x2672,  0x1611,  0x0630,  0x76d7,  0x66f6,  0x5695,  0x46b4,
    0xb75b,  0xa77a,  0x9719,  0x8738,  0xf7df,  0xe7fe,  0xd79d,  0xc7bc,
    0x48c4,  0x58e5,  0x6886,  0x78a7,  0x0840,  0x1861,  0x2802,  0x3823,
    0xc9cc,  0xd9ed,  0xe98e,  0xf9af,  0x8948,  0x9969,  0xa90a,  0xb92b,
    0x5af5,  0x4ad4,  0x7ab7,  0x6a96,  0x1a71,  0x0a50,  0x3a33,  0x2a12,
    0xdbfd,  0xcbdc,  0xfbbf,  0xeb9e,  0x9b79,  0x8b58,  0xbb3b,  0xab1a,
    0x6ca6,  0x7c87,  0x4ce4,  0x5cc5,  0x2c22,  0x3c03,  0x0c60,  0x1c41,
    0xedae,  0xfd8f,  0xcdec,  0xddcd,  0xad2a,  0xbd0b,  0x8d68,  0x9d49,
    0x7e97,  0x6eb6,  0x5ed5,  0x4ef4,  0x3e13,  0x2e32,  0x1e51,  0x0e70,
    0xff9f,  0xefbe,  0xdfdd,  0xcffc,  0xbf1b,  0xaf3a,  0x9f59,  0x8f78,
    0x9188,  0x81a9,  0xb1ca,  0xa1eb,  0xd10c,  0xc12d,  0xf14e,  0xe16f,
    0x1080,  0x00a1,  0x30c2,  0x20e3,  0x5004,  0x4025,  0x7046,  0x6067,
    0x83b9,  0x9398,  0xa3fb,  0xb3da,  0xc33d,  0xd31c,  0xe37f,  0xf35e,
    0x02b1,  0x1290,  0x22f3,  0x32d2,  0x4235,  0x5214,  0x6277,  0x7256,
    0xb5ea,  0xa5cb,  0x95a8,  0x8589,  0xf56e,  0xe54f,  0xd52c,  0xc50d,
    0x34e2,  0x24c3,  0x14a0,  0x0481,  0x7466,  0x6447,  0x5424,  0x4405,
    0xa7db,  0xb7fa,  0x8799,  0x97b8,  0xe75f,  0xf77e,  0xc71d,  0xd73c,
    0x26d3,  0x36f2,  0x0691,  0x16b0,  0x6657,  0x7676,  0x4615,  0x5634,
    0xd94c,  0xc96d,  0xf90e,  0xe92f,  0x99c8,  0x89e9,  0xb98a,  0xa9ab,
    0x5844,  0x4865,  0x7806,  0x6827,  0x18c0,  0x08e1,  0x3882,  0x28a3,
    0xcb7d,  0xdb5c,  0xeb3f,  0xfb1e,  0x8bf9,  0x9bd8,  0xabbb,  0xbb9a,
    0x4a75,  0x5a54,  0x6a37,  0x7a16,  0x0af1,  0x1ad0,  0x2ab3,  0x3a92,
    0xfd2e,  0xed0f,  0xdd6c,  0xcd4d,  0xbdaa,  0xad8b,  0x9de8,  0x8dc9,
    0x7c26,  0x6c07,  0x5c64,  0x4c45,  0x3ca2,  0x2c83,  0x1ce0,  0x0cc1,
    0xef1f,  0xff3e,  0xcf5d,  0xdf7c,  0xaf9b,  0xbfba,  0x8fd9,  0x9ff8,
    0x6e17,  0x7e36,  0x4e55,  0x5e74,  0x2e93,  0x3eb2,  0x0ed1,  0x1ef0
    };

WORD updcrc(WORD    icrc,
            BYTE   *icp,
            size_t  icnt)
{
      register WORD crc = icrc;
      register BYTE *cp = icp;
      register size_t cnt = icnt;

      while( cnt-- )
            crc = (crc<<B) ^ crctab[(crc>>(W-B)) ^ *cp++];

      return( crc );
}

#ifdef MAKETAB
main()
{
      initcrctab();
}

initcrctab()
{
      register  b, v, i;

      for( b = 0; b <= (1<<B)-1; ++b )
      {
            for( v = b<<(W-B), i = B; --i >= 0; )
                  v = v&0x8000 ? (v<<1)^P : v<<1;
            crctab[b] = v;

            printf( "0x%04x,", v & 0xFFFF );
            if( (b&7) == 7 )
                  printf("\n" );
            else  printf("  ");
```

```
101 |       }
102 | }
103 | #endif
104 |
105 | #ifdef TEST
106 |
107 | #include <stdio.h>
108 | #if defined(MSDOS) || defined(__MSDOS__)
109 |  #include "unistd.h"
110 | #else
111 |  #include <unistd.h>
112 | #endif
113 | #include <fcntl.h>
114 |
115 | #define MAXBUF 4096
116 |
117 | main(int argc, char **argv)
118 | {
119 |       int fd = 0;
120 |       int nr;
121 |       char buf[MAXBUF];
122 |       WORD crc;
123 |
124 |       if( argc > 1 )
125 |       {
126 |             if( (fd = open( argv[1], O_RDONLY )) < 0 )
127 |             {
128 |                   perror( argv[1] );
129 |                   exit( -1 );
130 |             }
131 |       }
132 |       crc = 0;
133 |       while( (nr = read( fd, buf, MAXBUF )) > 0 )
134 |             crc = updcrc( crc, (BYTE *)buf, nr );
135 |       printf( "%04x\n", crc );
136 |       if( nr != 0 )
137 |             perror( "reading" );
138 |       return 0;
139 | }
140 |
141 | #endif /* TEST */
```

TEXT STATISTICS

```
  4830 characters
   141 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
     7 constants [string]
   281 constants [numeric]
```

SYMBOL TABLE

```
    B       30      32      76+     91      93+
    BYTE            68      72     134
    MAKETAB         81
    MAXBUF         115     121     133
    MSDOS          108
    O_RDONLY       126
    P       24      94
    TEST           105
    W       27      76      93
    WORD            32      67+     71     122
    __MSDOS__      108
    argc           117     124
    argv           117     126     128
    b       89      91+     93      95      98
    buf            121     133     134
    cnt             73      75
    cp              72      76
    crc             71      76+     78     122     132     134+    135
    crctab          32      76      95
    defined        108+
    exit           129
    fcntl          113
    fd             119     126     133
    h      107     111     113
    i       89      93+
    icnt            69      73
    icp             68      72
    icrc            67      71
    initcrctab              84      87
    main            82     117
    nr             120     133     134     136
    open           126
    perror         128     137
    printf          97      99     100     135
    read           133
    size_t          69      73
    stdio          107
    unistd         111
    updcrc          67     134
    v       89      93      94+     95      97
```

```
/* +++Date last modified: 05-Jul-1997 */

/* Crc - 32 BIT ANSI X3.66 CRC checksum files */

#include <stdio.h>
#include "crc.h"

#ifdef __TURBOC__
 #pragma warn -cln
#endif

/************************************************************************\
|* Demonstration program to compute the 32-bit CRC used as the frame  *|
|* check sequence in ADCCP (ANSI X3.66, also known as FIPS PUB 71     *|
|* and FED-STD-1003, the U.S. versions of CCITT's X.25 link-level     *|
|* protocol).  The 32-bit FCS was added via the Federal Register,     *|
|* 1 June 1982, p.23798.  I presume but don't know for certain that   *|
|* this polynomial is or will be included in CCITT V.41, which        *|
|* defines the 16-bit CRC (often called CRC-CCITT) polynomial.  FIPS  *|
|* PUB 78 says that the 32-bit FCS reduces otherwise undetected       *|
|* errors by a factor of 10^-5 over 16-bit FCS.                       *|
\************************************************************************/

/* Need an unsigned type capable of holding 32 bits; */

typedef DWORD UNS_32_BITS;

/* Copyright (C) 1986 Gary S. Brown.  You may use this program, or
   code or tables extracted from it, as desired without restriction.*/

/* First, the polynomial itself and its table of feedback terms.  The  */
/* polynomial is                                                       */
/* X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X^1+X^0 */
/* Note that we take it "backwards" and put the highest-order term in  */
/* the lowest-order bit.  The X^32 term is "implied"; the LSB is the   */
/* X^31 term, etc.  The X^0 term (usually shown as "+1") results in    */
/* the MSB being 1.                                                    */

/* Note that the usual hardware shift register implementation, which   */
/* is what we're using (we're merely optimizing it by doing eight-bit  */
/* chunks at a time) shifts bits into the lowest-order term.  In our   */
/* implementation, that means shifting towards the right.  Why do we   */
/* do it this way?  Because the calculated CRC must be transmitted in  */
/* order from highest-order term to lowest-order term.  UARTs transmit */
/* characters in order from LSB to MSB.  By storing the CRC this way,  */
/* we hand it to the UART in the order low-byte to high-byte; the UART */
/* sends each low-bit to hight-bit; and the result is transmission bit */
/* by bit from highest- to lowest-order term without requiring any bit */
/* shuffling on our part.  Reception works similarly.                  */

/* The feedback terms table consists of 256, 32-bit entries.  Notes:   */
/*                                                                     */
/*  1. The table can be generated at runtime if desired; code to do so */
/*     is shown later.  It might not be obvious, but the feedback      */
/*     terms simply represent the results of eight shift/xor opera-    */
/*     tions for all combinations of data and CRC register values.     */
/*                                                                     */
/*  2. The CRC accumulation logic is the same for all CRC polynomials, */
/*     be they sixteen or thirty-two bits wide.  You simply choose the */
/*     appropriate table.  Alternatively, because the table can be     */
/*     generated at runtime, you can start by generating the table for */
/*     the polynomial in question and use exactly the same "updcrc",   */
/*     if your application needn't simultaneously handle two CRC       */
/*     polynomials.  (Note, however, that XMODEM is strange.)          */
/*                                                                     */
/*  3. For 16-bit CRCs, the table entries need be only 16 bits wide;   */
/*     of course, 32-bit entries work OK if the high 16 bits are zero. */
/*                                                                     */
/*  4. The values must be right-shifted by eight bits by the "updcrc"  */
/*     logic; the shift must be unsigned (bring in zeroes).  On some   */
/*     hardware you could probably optimize the shift in assembler by  */
/*     using byte-swap instructions.                                   */

static UNS_32_BITS crc_32_tab[] = { /* CRC polynomial 0xedb88320 */
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
```

```
0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

DWORD updateCRC32(unsigned char ch, DWORD crc)
{
      return UPDC32(ch, crc);
}

Boolean_T crc32file(char *name, DWORD *crc, long *charcnt)
{
      register FILE *fin;
      register DWORD oldcrc32;
      register int c;

      oldcrc32 = 0xFFFFFFFF; *charcnt = 0;
#ifdef MSDOS
      if ((fin=fopen(name, "rb"))==NULL)
#else
      if ((fin=fopen(name, "r"))==NULL)
#endif
      {
            perror(name);
            return Error_;
      }
      while ((c=getc(fin))!=EOF)
      {
            ++*charcnt;
            oldcrc32 = UPDC32(c, oldcrc32);
      }

      if (ferror(fin))
      {
            perror(name);
            *charcnt = -1;
      }
      fclose(fin);

      *crc = oldcrc32 = ~oldcrc32;

      return Success_;
}

DWORD crc32buf(char *buf, size_t len)
{
      register DWORD oldcrc32;

      oldcrc32 = 0xFFFFFFFF;

      for ( ; len; --len, ++buf)
      {
            oldcrc32 = UPDC32(*buf, oldcrc32);
      }

      return ~oldcrc32;

}

#ifdef TEST

main(int argc, char *argv[])
{
      DWORD crc;
      long charcnt;
      register errors = 0;

      while(--argc > 0)
      {
            errors |= crc32file(*++argv, &crc, &charcnt);
            printf("%08lX %7ld %s\n", crc, charcnt, *argv);
      }
      return(errors != 0);
}

#endif /* TEST */
```

```
TEXT STATISTICS

  8780 characters
   190 lines

LEXICAL STATISTICS

    47 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     4 constants [string]
   263 constants [numeric]

SYMBOL TABLE

    Boolean_T      125
    DWORD           26     120+    125     128     159     161     178
    EOF            141
    Error_         139
    FILE           127
    MSDOS          132
    NULL           133     135
    Success_       156
    TEST           174
    UNS_32_BITS             26      74
    UPDC32         122     144     167
    __TURBOC__                8
    argc           176     182
    argv           176     184     185
    buf            159     165     167
    c      129     141     144
    ch             120     122
    charcnt        125     131     143     150     179     184     185
    cln              9
    crc            120     122     125     154     178     184     185
    crc32buf       159
    crc32file      125     184
    crc_32_tab              74
    errors         180     184     187
    fclose         152
    ferror         147
    fin            127     133     135     141     147     152
    fopen          133     135
    getc           141
    h        5
    len            159     165+
    main           176
    name           125     133     135     138     149
    oldcrc32       128     131     144+    154+    161     163     167+    170
    perror         138     149
    printf         185
    size_t         159
    stdio            5
    updateCRC32              120
    warn             9
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * Donated to public domain
 *
 * Designation:  CSplit
 *
 * Description:  This program is used to process source files for
 *               the purpose of transmission through a FidoNet (tm)
 *               echo in such a manner as to circumvent over-size
 *               message problems.
 *
 *  This program implements the following specifications:
 *
 *  1) a. Combine multiple source files.
 *     b. Split source into 90 line sections.
 *     c. Add header and trailer marker lines delimiting each
 *        section and file.
 *  2) a. Delete any trailing whitespace.
 *     b. Replace tabs with spaces honoring tabstop boundaries.
 *     c. Default to 4 columns per tabstop.
 *     d. Allow user to specify alternate tabstop column number.
 *  3) a. Wrap lines longer than 75 characters long using the C
 *        "\ at the end of a line" notation (using space break).
 *     b. Distinguish wrapped lines from user-continued lines by
 *        inserting a line with a single "\" character between the
 *        two lines that contain the wrapped text.
 *  4) a. Calculate a CRC for each section and include it in the
 *        section trailer marker lines.
 *  5) a. Provide a help display for program usage when the
 *        program is executed without parameters.
 *  6) a. Provide as detailed of explanation as possible when
 *        an unexpected condition occurs.
 *     b. Attempt to continue execution for all but the most severe
 *        errors.
 *
 *
 * Syntax:
 *
 * Split:     CSPLIT  [/tn] [/wn] [/ln] [/sc]  outfile  src.ext [ ... ]
 *
 * Extract:   CSPLIT  /x  infile  [ ... ]
 *
 * Where:      /t n - For TAB character expansion, the number of columns
 *                    for each TAB stop.  (defaults to every 4 columns)
 *             /w n - For width control, the column number to be used for
 *                    breaking long lines.  (the default is column 75)
 *             /l n - For length control, the number of lines to limit
 *                    each section or 0 for no split.  (default is 90)
 *             /s c - Use 'c' as an alternate line separator character
 *                    instead of the default separator character, '>'.
 *                    Ignored for extraction - matches separator found
 *                    in extract file.  However, the extract file must
 *                    only use one separator character.
 *           infile - Input file name.  An extension indicates that the
 *                    file contains the sections in proper consecutive
 *                    order ready for extraction.  Otherwise, infile.001,
 *                    infile.002, etc., will be used.
 *          outfile - Name of the output file(s).  The extension will
 *                    be ignored if specified and each output file will
 *                    be named such that the section number will be the
 *                    extension (e.g., outfile.001, outfile.002, etc..)
 *          src.ext - The first source file..etc  Wildcard filespecs are
 *                    supported only under non-ANSI compiling conditions.
 *
 * Notes:  Paths are supported for all filenames, however, the paths
 *         are not preserved internally during the split operation.
 *         The extraction process will therefore only create files in
 *         the current directory.
 *
 * Revision History:  ( thanks to all the contributors )
 *
 * 08/31/93  Fred Cole  Original draft
 * 09/05/93  Fred Cole  Added CRC calculation and extraction ability.
 *                      Fixed a line wrap problem.
 * 09/14/93  Fred Cole  Added conditional compilation directives to
 *                      allow non-ANSI filespec support.  Squashed an
 *                      extract() function bug.
 * 11/21/93  Thad Smith Test for incomplete input file on extraction.
 *                      Remove spaces added in message transmission.
 *                      Default to 90 lines/section.  Fix tab expansion.
 * 12/03/93  Fred Cole  Fixed a cleanup() function bug.
 * 12/09/93  Keith Campbell / TS
 *                      Fixed bug with options preceded by '-' and fixed
 *                      tempfile opening bug.
 * 01/02/94  David Nugent / FC
 *                      Additions for findfirst/findnext support for
 *                      MSC6 (& 7) and OS/2 in initlist() routine and
 *                      portable.h header file.
 * 01/02/94  Auke Reitsma / FC
 *                      Increased number of chars read in to prevent line
 *                      numbers from becoming out-of-sync with input.
 * 01/12/94  Chad Wagner / FC
 *                      Correction to initlist() function to eliminate
 *                      redundant line increment.
 *--- v2.0 ---------------------------------------------------------------
 * 07/23/94  Keith Campbell / FC
 *                      Modified to not abort extraction when a CRC
 *                      mismatch occurs - just issue warning message.
 * 07/30/94  Auke Reitsma / FC
```

```
101 |  *                      Added multiple file extraction ability.
102 |  * 09/17/94  Keith Campbell / FC
103 |  *                      Added version separator lines.
104 |  * 10/28/94  Bob Stout / FC
105 |  *                      Added separator character, width and length
106 |  *                      command line options.
107 |  * 12/18/94  Fred Cole  Revised code to facilitate maintenance.
108 |  * 12/27/94  Fred Cole  Limited the minimum width for breaking long
109 |  *                      lines to column 55 since this is the length
110 |  *                      of the longest separator line.
111 |  * 01/15/95  David Gersic / FC
112 |  *                      Modified the line wrap logic to handle long
113 |  *                      sequences of characters lacking whitespace.
114 |  *--- v2.1 ---------------------------------------------------------------
115 |  * 10/30/95  Phi Nguyen / FC
116 |  *                      Added file extraction messages.
117 |  * 10/31/95  Fred Cole  Added ability to extract unconcatenated files.
118 |  *                      Added path support except for extracted files
119 |  *                      ( i.e., paths are not preserved internally ).
120 |  * 11/06/95  Fred Cole  Corrected tabstop calculation on wrapped lines.
121 |  * 11/07/95  Fred Cole  Increased max section length to SHRT_MAX lines.
122 |  * 11/08/95  Bob Stout / FC
123 |  *                      Disable the split logic when a 0 section length
124 |  *                      is specified with the /L command line option.
125 |  *--- v2.2 ---------------------------------------------------------------
126 |  * 11/22/95  Fred Cole  Modified logic to ignore leading whitespace added
127 |  *                      to separator lines ( mail reader quoting? ).
128 |  * 11/22/95  Doug Nazar (DN2) / FC
129 |  *                      Modified sscanf() format specifiers to correctly
130 |  *                      convert CRC values for 32 bit platforms.
131 |  * 11/22/95  Fred Cole  Changed TRUE/FALSE enum to macros to accommodate
132 |  *                      systems where these identifiers already exist.
133 |  * 11/29/95  Bob Stout / FC
134 |  *                      Added unsigned casts to allow signed 'length' to
135 |  *                      be compared to SHRT_MAX on 16-bit platforms.
136 |  * 11/29/95  Doug Nazar (DN2) / FC
137 |  *                      Added setbuf() statement to unbuffer stdout.
138 |  * 06/02/96  Fred Cole  Renamed TRUE/FALSE macros to avoid possible
139 |  *                      identifier conflicts.
140 |  * 06/02/96  Darin McBride / FC
141 |  *                      Modified logic to allow source files located on
142 |  *                      on another drive to be processed.  This change
143 |  *                      assumes that ':' is not a valid character for a
144 |  *                      file name.
145 |  * 06/02/96  Fred Cole  Corrected error in extract logic when file does
146 |  *                      not exist.
147 |  */
148 |
149 | #include <ctype.h>
150 | #include <stdio.h>
151 | #include <stdlib.h>
152 | #include <string.h>
153 | #include <limits.h>
154 | #include "csplit.h"
155 |
156 | char tempfile[MAXFSPEC + 1]; /* necessary evils - global variables */
157 | FILE *finp = NULL;
158 | FILE *fout = NULL;
159 | FILE *ftmp = NULL;
160 | SLST *head = NULL;
161 | SLST *cur  = NULL;
162 |
163 | int main (int argc, char *argv[])
164 | {
165 |   char     sepchar = SEP_CDEF;
166 |   char     outfile[MAXFSPEC + 1];
167 |   char    *sptr    = 0;
168 |   int      argndx  = 1;
169 |   int      chr     = 0;
170 |   int      extract = B_FALSE;
171 |   int      length  = LENDEF;
172 |   int      retc    = 0;
173 |   int      tabstop = TABDEF;
174 |   int      width   = WIDDEF;
175 |
176 |
177 |   setbuf (stdout, 0);                /* DN2: buffered output fix */
178 |   printf ("\nCSplit %s  (pd) 1993-1996 by Fred Cole\n", VERSION);
179 |   printf ("This executable program is public domain.\n\n");
180 |
181 |   if (1 == argc)
182 |   {
183 |     disp_help ();
184 |     return SYNTAX;
185 |   }
186 |
187 |   while (('/' == argv[argndx][0]) || ('-' == argv[argndx][0]))
188 |   {
189 |     chr = toupper (argv[argndx][1]);
190 |
191 |     switch (chr)
192 |     {
193 |       case '?':
194 |       case 'H':                  /* /H,/? help option */
195 |         disp_help ();
196 |         return SYNTAX;
197 |
198 |       case 'X':                  /* /X extract option */
199 |         extract = B_TRUE;
200 |         break;
```

```
201 |
202 |       case 'T':                  /* /T tab option */
203 |         tabstop = atoi (&argv[argndx][2]);
204 |
205 |         if ((tabstop < TABMIN) || (tabstop > TABMAX))
206 |           printf ("Invalid tab parameter \"%s\" (%d-%d).\n", argv[argndx], TABMIN, TABMAX);
207 |
208 |         if (tabstop < TABMIN)
209 |           tabstop = TABMIN;
210 |         else if (tabstop > TABMAX)
211 |           tabstop = TABMAX;
212 |
213 |         break;
214 |
215 |       case 'W':                  /* /W width option */
216 |         width = atoi (&argv[argndx][2]);
217 |
218 |         if ((width < WIDMIN) || (width > WIDMAX))
219 |           printf ("Invalid width parameter \"%s\" (%d-%d).\n", argv[argndx], WIDMIN, WIDMAX);
220 |
221 |         if (width < WIDMIN)
222 |           width = WIDMIN;
223 |         else if (width > WIDMAX)
224 |           width = WIDMAX;
225 |
226 |         break;
227 |
228 |       case 'L':                  /* /L length option */
229 |         if (('0' == argv[argndx][2]) && ('\0' == argv[argndx][3]))
230 |           length = 0;
231 |         else
232 |         {
233 |           length = atoi (&argv[argndx][2]);
234 |
235 |           if ((length < LENMIN) || ((unsigned)length > LENMAX))
236 |             printf ("Invalid length parameter \"%s\" (0,%d-%d).\n", argv[argndx], LENMIN, LENMAX);
237 |
238 |           if (length < LENMIN)
239 |             length = LENDEF;
240 |           else if ((unsigned)length > LENMAX)
241 |             length = LENMAX;
242 |         }
243 |
244 |         break;
245 |
246 |       case 'S':                  /* /S separator character option */
247 |         sscanf (&argv[argndx][2], "%c", &sepchar);
248 |
249 |         if (0 == isgraph (sepchar))
250 |         {
251 |           printf ("Invalid input parameter \"%s\".\n", argv[argndx]);
252 |           sepchar = SEP_CDEF;
253 |         }
254 |         break;
255 |
256 |       default:
257 |         printf ("Ignoring unknown input parameter \"%s\".\n", argv[argndx]);
258 |     }
259 |     ++argndx;
260 |   }
261 |
262 |   if (B_TRUE == extract)
263 |   {
264 |     if (argndx == argc)
265 |     {
266 |       printf ("No file argument specified for extraction.\n");
267 |       return SYNTAX;
268 |     }
269 |
270 |     /* AR: handle multiple files; break on error */
271 |     for ( ; argndx < argc; ++argndx)
272 |       if (NOERR != (retc = extr_file (argv[argndx], sepchar)))
273 |         break;
274 |
275 |     cleanup ();
276 |     return retc;
277 |   }
278 |
279 |   if ((argc - argndx) < 2)
280 |   {
281 |     printf ("Missing input and/or output file name arguments.\n");
282 |     disp_help ();
283 |     return SYNTAX;
284 |   }
285 |
286 |   if (NULL != (sptr = strrchr (argv[argndx], '\\')))  /* ignore path */
287 |   {
288 |     if (NULL != (sptr = strchr (sptr, '.')))
289 |       *sptr = '\0';                              /* truncate any ext */
290 |   }
291 |   else if (NULL != (sptr = strchr (argv[argndx], '.')))
292 |   {
293 |     *sptr = '\0';                                /* truncate any ext */
294 |   }
295 |
296 |   if (strlen (argv[argndx]) > MAXFSPEC)
297 |   {
298 |     printf ("Output file name argument too long.\n");
299 |     return SYNTAX;
300 |   }
```

```
301 |
302 |   strncpy (outfile, argv[argndx], MAXFSPEC);
303 |   outfile[MAXFSPEC] = '\0';                     /* ensure termination */
304 |
305 |   if (NOERR != (retc = init_list (argc, argv, ++argndx)))
306 |   {
307 |     cleanup ();
308 |     return retc;
309 |   }
310 |
311 |   retc = split_src (head, outfile, length, width, tabstop, sepchar);
312 |   cleanup ();
313 |   return retc;
314 | }
315 |
316 | /*
317 |  * add_list - Add a file name to linked list of files to be processed.
318 |  */
319 | SLST *add_list (char *fname)
320 | {
321 |   SLST *new = NULL;
322 |
323 |   if (NULL == (new = (SLST *)malloc (sizeof(SLST))))
324 |   {
325 |     puts ("Error allocating memory.\n");
326 |   }
327 |   else
328 |   {
329 |     strcpy (new->srcfile, fname);
330 |     new->next = NULL;
331 |
332 |     if (NULL == cur)
333 |       head = new;
334 |     else
335 |       cur->next = new;
336 |   }
337 |   cur = new;
338 |   return cur;
339 | }
340 |
341 | /*
342 |  * cleanup - Just a convenient way to provide centralized housekeeping.
343 |  */
344 | void cleanup (void)
345 | {
346 |   free_list ();
347 |
348 |   if (NULL != finp)  fclose (finp);
349 |   if (NULL != fout)  fclose (fout);
350 |   if (NULL != ftmp)  fclose (ftmp);
351 |
352 |   /* Now, does it really exist? */
353 |   if (NULL != (ftmp = fopen (tempfile, "r")))
354 |   {
355 |     fclose (ftmp);
356 |     remove (tempfile);
357 |   }
358 | }
359 |
360 | /*
361 |  * csp_fgets - A custom fgets() function that expands
362 |  *             tabs, deletes trailing whitespace and
363 |  *             performs line wrapping.
364 |  */
365 | char *csp_fgets (char *s, int len, FILE * fp, int tabstop)
366 | {
367 |   static char  sbuf[LLENMAX * 2]; /* big enough for TAB expansion */
368 |   static char *beg  = sbuf;
369 |   static int   tofs = 0;
370 |   static int   wrap = B_FALSE;
371 |
372 |   char *e = 0;
373 |   char *w = 0;
374 |   char *p = s;
375 |   char *q = 0;
376 |
377 |   int   ch     = 0;
378 |   int   cnt    = 0;
379 |   int   i      = 0;
380 |   int   spaces = 0;
381 |
382 |   if (B_TRUE == wrap)             /* if line wrap */
383 |   {
384 |     tofs += (int)(beg - sbuf);    /* adj. TAB column offset */
385 |     strcpy (s, "\\\n");
386 |     memmove (sbuf, beg, strlen (beg) + 1); /* DG: Modification for  */
387 |     beg  = sbuf;                           /* DG: long lines w/o WS */
388 |     wrap = B_FALSE;
389 |     return s;
390 |   }
391 |
392 |   while ((cnt < len-1) && ('\n' != ch))
393 |   {                                 /* get next char from buffer */
394 |     if (0 == (ch = *beg++))         /* if buffer empty */
395 |     {
396 |       memset (sbuf, 0, sizeof (sbuf));
397 |       beg = fgets (sbuf, LLENMAX, fp); /* grab another string */
398 |
399 |       if (NULL == beg)              /* if end of file... */
400 |       {
```

```
401 |         beg = sbuf;
402 |         *beg = 0;
403 |
404 |         if (0 == cnt)
405 |           return NULL;                /*  and buffer empty */
406 |       }
407 |       else
408 |       {
409 |         w = e = &sbuf[i = strlen (sbuf)]; /* find 1st trailing ws char */
410 |
411 |         while ((w > sbuf) && (isspace (*(w - 1))))
412 |           --w;
413 |
414 |         if (('\n' == *(e - 1)) || /* if terminated w/newline char */
415 |             (i < (len-1)))        /* or unterminated short line */
416 |         {
417 |           *w++ = '\n';            /* terminate with newline char */
418 |         }
419 |
420 |         *w = 0;
421 |         ch = *beg++;
422 |       }
423 |     }
424 |
425 |     if (ch == '\t')               /* if TAB character */
426 |     {                             /* space to next tab stop */
427 |       /*
428 |        * TS: The following code has been changed to pad to the next
429 |        * tab stop, rather than insert a fixed number of spaces.
430 |        */
431 |       spaces = tabstop - (((int)(tofs + beg - sbuf) - 1) % tabstop);
432 |       memmove (beg + spaces - 1, beg, strlen (beg) + 1);
433 |
434 |       for (q = beg - 1; spaces > 0; --spaces)
435 |         *q++ = ' ';
436 |
437 |       ch = ' ';                   /* change TAB to space */
438 |     }
439 |
440 |     *p++ = (char)ch;              /* update output string */
441 |     ++cnt;
442 |
443 |     if ((cnt == len - 1) && ('\n' != ch))   /* if need to wrap line */
444 |     {
445 |       beg -= 2;                   /* make room for "\\\n" characters */
446 |       e = beg;
447 |       p -= 2;
448 |       q = p;
449 |
450 |       /* unget characters to 1st previous space */
451 |       while ((e > sbuf) && (' ' != *(e - 1)))
452 |       {
453 |         --q;
454 |         --e;
455 |       }
456 |
457 |       /*if (((beg - e) < len) && (e != sbuf))*/
458 |       if (e != sbuf)              /* if wrap on space char */
459 |       {                           /* ( else wrap asis ) */
460 |         p = q;
461 |         beg = e;
462 |       }
463 |
464 |       *p++ = '\\';               /* terminate current line */
465 |       *p++ = '\n';
466 |       wrap = B_TRUE;             /* flag wrap line pending */
467 |     }
468 |   } /*end_while*/
469 |
470 |   if ('\n' == ch)
471 |     tofs = 0;
472 |
473 |   *p = 0;
474 |   return s;
475 | }
476 |
477 | /*
478 |  * disp_help - Display help screen to user.
479 |  */
480 | void disp_help (void)
481 | {
482 |   puts ("This utility processes C/C++ source files for the purpose of transmission");
483 |   puts ("through a FidoNet (tm) echo so as to circumvent oversize message problems.\n");
484 |   puts ("Split:    CSplit  [ [/tn] [/wn] [/ln] [/sc] ]  outfile  srcfile [ ... ]");
485 |   puts ("Extract:  CSplit  /x  infile  [ ... ]\n");
486 |   puts ("Where:      /t n - For TAB character expansion, the number of columns");
487 | printf ("                   for each TAB stop.  (min/default/max: %d/%d/%d)\n", TABMIN, TABDEF, TABMAX
    |  );
488 |   puts ("            /w n - For width control, the column number to be used for");
489 | printf ("                   breaking long lines.  (min/default/max: %d/%d/%d)\n", WIDMIN, WIDDEF, WIDM
    |  AX);
490 |   puts ("            /l n - For length control, the number of lines to limit");
491 | printf ("                   each section.  (0 disable, min/default/max: %d/%d/%d)\n", LENMIN, LENDEF,
    |  LENMAX);
492 |   puts ("            /s c - Use 'c' as the line separator character instead");
493 | printf ("                   of the default separator character, '%c'.\n", SEP_CDEF);
494 |   puts ("         outfile - Output file name.  The extension will be the");
495 |   puts ("                   sequential section number.");
496 | #if defined(__STDC__)
497 |   puts ("         srcfile - Source file(s)  (no wildcard support)");
```

```
#else
  puts ("         srcfile - Source file(s)  (wildcards supported)");
#endif
  puts ("          infile - Input file name.  An extension indicates file contains the");
  puts ("                   sections in proper consecutive order ready for extraction.");
  puts ("                   Otherwise, infile.001, infile.002, etc., will be used.");
}

/*
 * extr_file - This function processes a properly concatenated file
 *             of consequtive sections to extract the original source.
 */
int extr_file (char *infile, char sepchar)
{
  static char  line [WIDMAX * 2];
  static char  line2[WIDMAX * 2];

  char         outfile[MAXFSPEC + 1];
  char         sep_ln[32];
  char        *sptr = 0;
  char         tchar      = 0;

  int          curpart    = 0;
  int          found      = B_FALSE;
  int          indx       = 0;
  int          in_section = B_FALSE;
  int          len        = 0;
  int          key        = 0;
  int          lines      = 0;
  int          maxpart    = 0;
  int          pos_wrap   = B_FALSE;
  int          sepmax     = 0;
  int          seppart    = 0;
  int          sep_id_len = 0;
  int          temp       = 0;

  unsigned short crc = 0;
  unsigned short sepcrc = 0;


  for (temp = 0; temp < SEP_CLEN; ++temp)
    sep_ln[temp] = (char)sepchar;

  sep_ln[temp++] = ' ';
  strcpy (&sep_ln[temp], SEP_ID);
  sep_id_len = strlen (sep_ln);

  if (NULL == (finp = fopen (infile, "r")))
  {
    char fname[MAXFSPEC + 1];

    if ((NULL == (sptr = strrchr (infile, '\\'))) && /* ignore path */
        (NULL == (sptr = strrchr (infile, ':'))))
    {
      sptr = infile;                        /* V2.2#3 correction */
    }

    if (NULL != strchr (sptr, '.'))         /* if extension exists */
    {
      printf ("Error opening input file for extraction.\n");
      return FILEIO;
    }

    if ((strlen (infile) + 4) > MAXFSPEC)   /* if file spec too large */
    {
      printf ("Input file name argument too long.\n");
      return SYNTAX;
    }

    indx = 1;
    sprintf (fname, "%s.%03d", infile, indx);

    if (NULL != (finp = fopen (fname, "r")))
    {
      sprintf (tempfile, "%s.CSP", infile);

      if (NULL == (ftmp = fopen (tempfile, "w")))
      {
        printf ("Unable to open \"%s\" temp file.\n", tempfile);
        return FILEIO;
      }

      printf ("Processing input files: %s ...\n", fname);

      do
      {
        while (NULL != fgets (line, sizeof (line), finp))
          fputs (line, ftmp);

        fclose (finp);
        ++indx;
        sprintf (fname, "%s.%03d", infile, indx);
        finp = fopen (fname, "r");

      } while (NULL != finp);

      fclose (ftmp);

      if (NULL == (finp = fopen (tempfile, "r")))
      {
```

```
598 |         printf ("Error opening temp file \"%s\" for extraction.\n", outfile);
599 |         return FILEIO;
600 |       }
601 |     }
602 |     else
603 |     {
604 |       printf ("Error opening input file \"%s\" for extraction.\n", infile);
605 |       return FILEIO;
606 |     }
607 |   }
608 |   else
609 |   {
610 |     printf ("Processing input file: %s\n", infile);
611 |   }
612 |
613 |   crc = curpart = maxpart = lines = 0;
614 |   in_section = pos_wrap = B_FALSE;
615 |   fout = NULL;
616 |   initcrctab ();
617 |   *line2 = 0;
618 |
619 |   while (NULL != finp)
620 |   {
621 |     /* AR: increased line input size */
622 |     sptr = fgets (line, sizeof (line), finp);
623 |
624 |     if (NULL == sptr)
625 |     {
626 |       if (feof (finp))            /* end of file */
627 |       {
628 |         fclose (finp);
629 |         finp = NULL;
630 |       }
631 |       else
632 |       {
633 |         if (lines)
634 |           printf ("(line %d) ", lines);
635 |
636 |         printf ("Unable to read from input file:  %s\n", infile);
637 |         return FILEIO;
638 |       }
639 |     }
640 |     else                          /* process line */
641 |     {
642 |       /* TS: eliminate any added trailing spaces */
643 |       for (indx = strlen (sptr) - 1; indx && (' ' == line[indx - 1]); --indx)
644 |         continue;
645 |
646 |       line[indx]   = '\n';
647 |       line[indx+1] = '\0';
648 |       ++lines;
649 |
650 |       /*
651 |        * Find 1st separator line to determine if correct separator
652 |        * character is being used and use the one that was found.
653 |        */
654 |       if (!found && (0 != strstr (line, SEP_ID)))
655 |       {
656 |         for (temp = 0; isspace (line[temp]); ++temp)
657 |           ;  /* null statement */
658 |
659 |         tchar = line[temp];               /* skip leading ws */
660 |
661 |         if (sepchar != tchar)
662 |           for (temp = 0; temp < SEP_CLEN; ++temp)
663 |             sep_ln[temp] = tchar;
664 |
665 |         found = B_TRUE;
666 |       }
667 |
668 |       if (0 != (sptr = strstr (line , sep_ln))) /* if separator line */
669 |       {
670 |         sptr += sep_id_len;
671 |
672 |         if (sptr == strstr (sptr, SEP_BF))  /* if begin file */
673 |         {
674 |           if (NULL != fout)
675 |           {
676 |             printf ("(line %d) ", lines);
677 |             puts ("Encountered 2nd \"Begin file\" separator");
678 |             puts ("line before expected \"End file\" separator line:");
679 |             puts (line);
680 |             return PROCESS;
681 |           }
682 |
683 |           sptr += strlen (SEP_BF);
684 |
685 |           if (1 != sscanf (sptr, "%s", outfile))
686 |           {
687 |             printf ("(line %d) ", lines);
688 |             puts ("Unable to parse filename from separator line:");
689 |             puts (line);
690 |             return PROCESS;
691 |           }
692 |
693 |           if (NULL != (fout = fopen (outfile, "r")))
694 |           {
695 |             key = 0;
696 |             printf ("\nOutput file already exists:  %s\n", outfile);
697 |
```

```
698 |             do
699 |             {
700 |               printf ("Overwrite? (y/n) ");
701 |               key = getchar ();
702 |               puts ("");
703 |               temp = key;
704 |
705 |               while (temp != '\n')
706 |                 temp = getchar();  /* eat any/all extra keystrokes */
707 |
708 |               if (('n' == key) || ('N' == key))
709 |                 return ABORT;
710 |
711 |             } while (('y' != key) && ('Y' != key));
712 |
713 |             if (NULL == freopen (outfile, "w", fout))
714 |             {
715 |               printf ("Unable to open file for output:  %s\n", outfile);
716 |               return FILEIO;
717 |             }
718 |           }
719 |           else
720 |           {
721 |             if (NULL == (fout = fopen (outfile, "w")))
722 |             {
723 |               printf ("Unable to open file for output:  %s\n", outfile);
724 |               return FILEIO;
725 |             }
726 |           }
727 |
728 |           printf ("Extracting file %s\n", outfile);   /* Phi Nguyen */
729 |           crc = updcrc (crc, (unsigned char *)line, strlen (line));
730 |         }
731 |         else if (sptr == strstr (sptr, SEP_EF))  /* if end file */
732 |         {
733 |           if (NULL == fout)
734 |           {
735 |             printf ("(line %d) ", lines);
736 |             puts ("Encountered an \"End file\" separator line");
737 |             puts ("before a \"Begin file\" separator line:");
738 |             puts (line);
739 |             return PROCESS;
740 |           }
741 |
742 |           if (fclose (fout))
743 |           {
744 |             printf ("Unable to close output file:  %s\n", outfile);
745 |             return FILEIO;
746 |           }
747 |
748 |           fout = NULL;
749 |           crc = updcrc (crc, (unsigned char *)line, strlen (line));
750 |         }
751 |         else if (sptr == strstr (sptr, SEP_BP))  /* if begin part */
752 |         {
753 |           if (B_TRUE == in_section)
754 |           {
755 |             printf ("(line %d) ", lines);
756 |             puts ("Encountered 2nd \"Begin part\" separator");
757 |             puts ("line before expected \"End part\" separator line:");
758 |             puts (line);
759 |             return PROCESS;
760 |           }
761 |           sptr += strlen (SEP_BP);
762 |
763 |           if (2 != sscanf (sptr, "%d/%d", &seppart, &sepmax))
764 |           {
765 |             printf ("(line %d) ", lines);
766 |             puts ("Unable to parse \"n/x\" from separator line:");
767 |             puts (line);
768 |             return PROCESS;
769 |           }
770 |
771 |           if (0 == maxpart)
772 |             maxpart = sepmax;
773 |
774 |           if (curpart+1 != seppart)
775 |           {
776 |             printf ("(line %d) ", lines);
777 |             printf ("Encountered part %d while expecting part %d:\n", seppart, curpart+1);
778 |             puts (line);
779 |             return PROCESS;
780 |           }
781 |           in_section = B_TRUE;
782 |           ++curpart;
783 |         }
784 |         else if (sptr == strstr (sptr, SEP_EP))  /* if end part */
785 |         {
786 |           if (B_FALSE == in_section)
787 |           {
788 |             printf ("(line %d) ", lines);
789 |             puts ("Encountered 2nd \"End part\" separator line");
790 |             puts ("before expected \"Begin part\" separator line:");
791 |             puts (line);
792 |             return PROCESS;
793 |           }
794 |           sptr += strlen (SEP_EP);
795 |           sptr = strstr (sptr, ": ");
796 |
797 |           if (1 != sscanf (sptr+2, "%hx", &sepcrc))   /* DN2: 32 bit fix */
```

```
798 |           {
799 |             printf ("(line %d) ", lines);
800 |             puts ("Corrupt CRC in separator line:");
801 |             puts (line);
802 |             return PROCESS;
803 |           }
804 |
805 |           if (crc != sepcrc)
806 |           {
807 |             printf ("(line %d) ", lines);
808 |             printf ("Calculated CRC mismatch (0x%04x):\n", crc);
809 |             puts (line);           /* KC: report CRC mismatch only */
810 |           }
811 |           crc = 0;
812 |           in_section = B_FALSE;
813 |
814 |           if (curpart == maxpart)               /* if finished */
815 |             break;
816 |         }
817 |         else
818 |         {
819 |           if (sptr != strstr (sptr, SEP_VR))     /* if not version */
820 |           {
821 |             printf ("(line %d) ", lines);
822 |             puts ("Unrecognized separator line:");
823 |             puts (line);
824 |             return PROCESS;
825 |           }
826 |         }
827 |       }
828 |       else                        /* else process data line */
829 |       {
830 |         if (B_TRUE == in_section) /* save only file data */
831 |         {
832 |           len = strlen (line);
833 |           crc = updcrc (crc, (unsigned char *)line, len);
834 |
835 |           if (B_TRUE == pos_wrap) /* if possible line wrap in progress */
836 |           {
837 |             if (0 == strncmp (line, "\\\n", 2))  /* if wrapped line */
838 |             {
839 |               strcpy (line, line2);
840 |               line[strlen (line) - 2] = 0;  /* remove wrap EOL */
841 |             }
842 |             else
843 |             {
844 |               strcat (line2, line);
845 |               strcpy (line, line2);
846 |             }
847 |             pos_wrap = B_FALSE;
848 |           }
849 |           else  if ('\\' == line[len - 2]) /* if possible wrapped line */
850 |           {
851 |             strcpy (line2, line);
852 |             pos_wrap = B_TRUE;
853 |           }
854 |
855 |           if ((B_FALSE == pos_wrap) &&
856 |               ((NULL == fout) || (EOF == fputs (line, fout))))
857 |           {
858 |             puts ("Error writing output\n");
859 |             return FILEIO;
860 |           }
861 |         }
862 |       }
863 |     }
864 |   } /*end_while*/
865 |
866 |   /* TS: Test for incompete processing. */
867 |   if (B_TRUE == in_section)
868 |   {
869 |     printf ("Error: end of input while processing section %d of %d\n", seppart, sepmax);
870 |     return PROCESS;
871 |   }
872 |
873 |   if (seppart != sepmax)
874 |   {
875 |     printf ("Error: end of input after processing section %d of %d\n", seppart, sepmax);
876 |     return PROCESS;
877 |   }
878 |   return NOERR;
879 | }
880 |
881 | /*
882 |  * free_list - This function simply frees each linked list item.
883 |  */
884 | void free_list (void)
885 | {
886 |   while (NULL != head)
887 |   {
888 |     cur = head->next;
889 |     free (head);
890 |     head = cur;
891 |   }
892 | }
893 |
894 | /*
895 |  * init_list - This function creates a linked list of input source
896 |  *             files.  Wildcard specifications are accommodated when
897 |  *             ANSI mode is not in effect.
```

```
 */
int init_list (int argc, char **argv, int argo)
{
  int i;
#if !defined(__STDC__)
  char         filename[MAXFSPEC + 1];
  char         path[PATHMAX + 1];
  char        *sptr;
  int          done;
  DOSFileData  fd;
#endif

  for (i = argo; i < argc; ++i)            /* process CL arguments */
  {
#if !defined(__STDC__)
    if (strlen (argv[i]) > (MAXFSPEC - FNAMELEN))
    {
      printf ("Input file argument too long:  %s\n", argv[i]);
      return SYNTAX;
    }

    done = FIND_FIRST(argv[i], 0x20, &fd);  /* david nugent */

    if (done)
    {
      printf ("Error with filespec: %s\n", argv[i]);
      return PROCESS;
    }

    strcpy (path, argv[i]);                /* preserve path */

    if (NULL != (sptr = strrchr (path, '\\')))
      *(sptr + 1) = '\0';
    else
      path[0] = '\0';

    while (!done)
    {
      if ('\0' != path[0])
      {
        strcpy (filename, path);

        if (NULL != (sptr = strrchr (ff_name(&fd), '\\')))
          strcat (filename, sptr + 1);      /* just in case */
        else
          strcat (filename, ff_name(&fd));
      }
      else
        strcpy (filename, ff_name(&fd));

      if (NULL == add_list (filename))      /* david nugent */
        return MEMORY;

      done = FIND_NEXT(&fd);
    }
    FIND_END(&fd);                          /* david nugent */
#else
    if (strlen (argv[i]) > MAXFSPEC)
    {
      printf ("Input file argument too long:  %s\n", argv[i]);
      return SYNTAX;
    }

    if (NULL == add_list (argv[i]))
      return MEMORY;

#endif
  }
  return NOERR;
}

/*
 * split_src - This function takes a linked list of input source
 *             files, concatenates the source and then splits it
 *             into sections of controlled size.
 */
int split_src (SLST *filelist, char *outfname, int length, int width, int tabstop, char sepchar)
{
  char           line[WIDMAX + 1];
  char          *ext     = 0;
  char           filename[MAXFSPEC + 1];
  char           outfile[MAXFSPEC + 1];
  char           sep_ln[32];
  char          *sptr    = 0;
  int            curpart = 0;
  int            key     = 0;
  int            lines   = 0;
  int            maxpart = 1;
  int            retc    = 0;
  int            temp    = 0;
  unsigned short crc     = 0;


  strcpy (outfile, outfname);

  if (NULL == (ext = strchr (outfile, '.')))     /* ignore any ext */
    ext = &outfile[strlen (outfile)];

  *ext = 0;                                      /* make temp file name */
  strcpy (tempfile, outfile);
```

```
 998 |   strcat (tempfile, ".CSP");
 999 |
1000 |   if (NULL == (ftmp = fopen (tempfile, "w")))
1001 |   {
1002 |     printf ("Error creating temp file:  %s\n", tempfile);
1003 |     return FILEIO;
1004 |   }
1005 |
1006 |   for (temp = 0; temp < 10; ++temp)
1007 |     sep_ln[temp] = (char)sepchar;
1008 |
1009 |   sep_ln[temp] = '\0';
1010 |
1011 |   for (cur = filelist, lines = 0; NULL != cur; cur = cur->next)
1012 |   {
1013 |     if (NULL == finp)
1014 |     {
1015 |       if (NULL == (finp = fopen (cur->srcfile, "r")))
1016 |       {
1017 |         printf ("Error opening source file:  %s\n", cur->srcfile);
1018 |         return FILEIO;
1019 |       }
1020 |
1021 |       if ((NULL != (sptr = strrchr (cur->srcfile, '\\'))) ||
1022 |           (NULL != (sptr = strrchr (cur->srcfile, ':')))) /* Darin McBride */
1023 |         strcpy (filename, sptr+1);
1024 |       else
1025 |         strcpy (filename, cur->srcfile);
1026 |
1027 |       retc = fprintf (ftmp, "%s %s%s%s %s\n", sep_ln, SEP_ID, SEP_BF, filename, sep_ln);
1028 |
1029 |       if (0 == retc)
1030 |       {
1031 |         puts ("Error writing output\n");
1032 |         return FILEIO;
1033 |       }
1034 |       ++lines;
1035 |     }
1036 |
1037 |     while (NULL != finp)
1038 |     {
1039 |       /*
1040 |        * The function csp_fgets() is equivalent to fgets() in that it
1041 |        * too reads n-1 characters or up to a newline character.  This
1042 |        * function additionally expands TAB characters, deletes trailing
1043 |        * whitespace and wraps lines exceeding the specified length.
1044 |        */
1045 |       if (NULL == csp_fgets (line, width, finp, tabstop))
1046 |       {
1047 |         if (feof (finp))
1048 |         {
1049 |           fclose (finp);
1050 |           finp = NULL;
1051 |           retc = fprintf (ftmp, "%s %s%s%s %s\n", sep_ln, SEP_ID, SEP_EF, filename, sep_ln);
1052 |
1053 |           if (0 == retc)
1054 |           {
1055 |             puts ("Error writing output\n");
1056 |             return FILEIO;
1057 |           }
1058 |           ++lines;      /* adjust line count */
1059 |         }
1060 |         else
1061 |         {
1062 |           puts ("Error reading input\n");
1063 |           return FILEIO;
1064 |         }
1065 |       }
1066 |       else
1067 |       {
1068 |         if (EOF == fputs (line, ftmp))
1069 |         {
1070 |           puts ("Error writing output\n");
1071 |           return FILEIO;
1072 |         }
1073 |         ++lines;
1074 |       }
1075 |     } /*end_while*/
1076 |   } /*end_for*/
1077 |
1078 |   if (0 != length)
1079 |   {
1080 |     /* There are 3 lines of overhead per section. */
1081 |     maxpart = lines / (length - 3);
1082 |
1083 |     if ((lines % (length - 3)) > 0)
1084 |       ++maxpart;                        /* for partial section */
1085 |   }
1086 |
1087 |   curpart = 1;
1088 |   sprintf (ext, ".%03d", curpart);     /* make 1st output file name */
1089 |
1090 |   /* warn user if the output filename is already in use */
1091 |   if (NULL != (fout = fopen (outfile, "r")))
1092 |   {
1093 |     key = 0;
1094 |     printf ("Output file already exists:  %s\n", outfile);
1095 |
1096 |     do
1097 |     {
```

```
1098 |       printf ("Overwrite? (y/n): ");
1099 |       key = (toupper) (getchar ());    /* prevent using toupper macro */
1100 |       puts ("");
1101 |       temp = key;
1102 |
1103 |       while (temp != '\n')
1104 |       {
1105 |         temp = getchar ();             /* eat all extra keystrokes */
1106 |       }
1107 |
1108 |       if ('N' == key)
1109 |         return ABORT;
1110 |
1111 |     } while ('Y' != key);
1112 |
1113 |     fclose (fout);
1114 |     fout = NULL;
1115 |   }
1116 |
1117 |   if (NULL == freopen (tempfile, "r", ftmp))
1118 |   {
1119 |     printf ("Error reopening temp file:  %s\n", tempfile);
1120 |     return FILEIO;
1121 |   }
1122 |
1123 |   initcrctab ();
1124 |
1125 |   while (NULL != ftmp)
1126 |   {
1127 |     lines = 0;
1128 |
1129 |     if (NULL == fout)
1130 |     {
1131 |       sprintf (ext, ".%03d", curpart); /* make output file name */
1132 |
1133 |       if (NULL == (fout = fopen (outfile, "w")))
1134 |       {
1135 |         printf ("Error opening output file:  %s\n", outfile);
1136 |         return FILEIO;
1137 |       }
1138 |
1139 |       retc = fprintf (fout, "%s %s%s%s %s\n", sep_ln, SEP_ID, SEP_VR, VERSION, sep_ln);
1140 |
1141 |       if (0 == retc)
1142 |       {
1143 |         puts ("Error writing output\n");
1144 |         return FILEIO;
1145 |       }
1146 |
1147 |       ++lines;
1148 |       retc = fprintf (fout, "%s %s%s%d/%d %s\n", sep_ln, SEP_ID, SEP_BP, curpart, maxpart, sep_ln);
1149 |
1150 |       if (0 == retc)
1151 |       {
1152 |         puts ("Error writing output\n");
1153 |         return FILEIO;
1154 |       }
1155 |
1156 |       ++lines;
1157 |     }
1158 |
1159 |     crc = 0;
1160 |
1161 |     while (((0 == length ) || (lines < (length - 1))) &&
1162 |            (NULL != ftmp))
1163 |     {
1164 |       if (NULL == fgets (line, WIDMAX, ftmp))
1165 |       {
1166 |         if (feof (ftmp))
1167 |         {
1168 |           fclose (ftmp);
1169 |           ftmp = NULL;
1170 |         }
1171 |         else
1172 |         {
1173 |           puts ("Error reading input\n");
1174 |           return FILEIO;
1175 |         }
1176 |       }
1177 |       else
1178 |       {
1179 |         crc = updcrc (crc, (unsigned char *)line, strlen (line));
1180 |
1181 |         if (EOF == fputs (line, fout))
1182 |         {
1183 |           puts ("Error writing output\n");
1184 |           return FILEIO;
1185 |         }
1186 |
1187 |         ++lines; /* increment line count */
1188 |       }
1189 |     } /*end_while*/
1190 |
1191 |     if (0 == fprintf (fout, "%s %s%s%d/%d crc: %04x %s\n", sep_ln, SEP_ID, SEP_EP, curpart, maxpart, c
     |     rc, sep_ln))
1192 |     {
1193 |       puts ("Error writing output\n");
1194 |       return FILEIO;
1195 |     }
1196 |
```

```
1197 |     fclose (fout);
1198 |     fout = NULL;
1199 |     ++curpart;
1200 |   } /*end_while*/
1201 |
1202 |   return NOERR;
1203 | }
1204 |
1205 |
1206 | /*
1207 |  * CRC-16f.c, from Snippets.  Calculate, intelligently, the CRC
1208 |  * of a dataset incrementally given a buffer full at a time.
1209 |  * Initialize crc to 0 for XMODEM, -1 for CCITT.
1210 |  */
1211 |
1212 | /*
1213 |  * P, the CRC polynomial, is used by XMODEM (almost CCITT).
1214 |  * If you change P, you must change crctab[]'s initial value
1215 |  * to what is printed by initcrctab().
1216 |  */
1217 | #define  P 0x1021
1218 | #define  W 16       /* number of bits in CRC: don't change it */
1219 | #define  B 8        /* number of bits per char: don't change it */
1220 |
1221 | unsigned short crctab[1<<B];
1222 |
1223 | void initcrctab (void)
1224 | {
1225 |   register b, v, i;
1226 |
1227 |   for (b = 0; b <= (1<<B) - 1; ++b)
1228 |   {
1229 |     for (v = b<<(W-B), i = B; --i >= 0; )
1230 |     {
1231 |       v = v & 0x8000 ? (v<<1)^P : v<<1;
1232 |     }
1233 |
1234 |     crctab[b] = v;
1235 |   }
1236 | }
1237 |
1238 | unsigned short updcrc (unsigned short icrc, unsigned char *icp, unsigned int icnt)
1239 | {
1240 |   register unsigned short crc = icrc;
1241 |   register unsigned char  *cp = icp;
1242 |   register unsigned int   cnt = icnt;
1243 |
1244 |   while (cnt--)
1245 |   {
1246 |     crc = (crc<<B) ^ crctab[(crc>>(W-B)) ^ *cp++];
1247 |   }
1248 |
1249 |   return crc;
1250 | }
1251 |
```

TEXT STATISTICS

37626 characters
1251 lines

LEXICAL STATISTICS

109 comments [std-C]
0 comments [C++]
17 preprocessor instructions
53 constants [character]
125 constants [string]
146 constants [numeric]

SYMBOL TABLE

```
ABORT          709    1109
B     1219    1221    1227    1229+   1246+
B_FALSE        170     370     388     521     523     528     614     786     812     847     855
B_TRUE         199     262     382     466     665     753     781     830     835     852     867
DOSFileData            907
EOF            856    1068    1181
FILE           157     158     159     365
FILEIO         558     577     599     605     637     716     724     745     859    1003    1018
      1032    1056    1063    1071    1120    1136    1144    1153    1174    1184    1194
FIND_END       953
FIND_FIRST             919
FIND_NEXT      951
FNAMELEN       913
LENDEF         171     239     491
LENMAX         235     236     240     241     491
LENMIN         235     236     238     491
LLENMAX        367     397
MAXFSPEC       156     166     296     302     303     515     547     561     903     913     955
       978     979
MEMORY         949     962
NOERR          272     305     878     966    1202
NULL           157     158     159     160     161     286     288     291     321     323     330
       332     348     349     350     353     399     405     545     549     550     555     570
       574     584     592     596     615     619     624     629     674     693     713     721
       733     748     856     886     929     940     948     961     993    1000    1011    1013
      1015    1021    1022    1037    1045    1050    1091    1114    1117    1125    1129    1133
      1162    1164    1169    1198
P     1217    1231
PATHMAX        904
PROCESS        680     690     739     759     768     779     792     802     824     870     876
       924
SEP_BF         672     683    1027
SEP_BP         751     761    1148
SEP_CDEF       165     252     493
SEP_CLEN       538     662
SEP_EF         731    1051
SEP_EP         784     794    1191
SEP_ID         542     654    1027    1051    1139    1148    1191
SEP_VR         819    1139
SLST           160     161     319     321     323+    974
SYNTAX         184     196     267     283     299     564     916     958
TABDEF         173     487
TABMAX         205     206     210     211     487
TABMIN         205     206     208     209     487
VERSION        178    1139
W     1218    1229    1246
WIDDEF         174     489
WIDMAX         218     219     223     224     489     512     513     976    1164
WIDMIN         218     219     221     222     489
__STDC__       496     902     912
add_list       319     948     961
argc           163     181     264     271     279     305     899     910
argndx         168     187+    189     203     206     216     219     229+    233     236     247
       251     257     259     264     271+    272     279     286     291     296     302     305
argo           899     910
argv           163     187+    189     203     206     216     219     229+    233     236     247
       251     257     272     286     291     296     302     305     899     913     915     919
       923     927     955     957     961
atoi           203     216     233
b     1225    1227+   1229    1234
beg            368     384     386+    387     394     397     399     401     402     421     431
       432+    434     445     446     461
ch             377     392     394     421     425     437     440     443     470
chr            169     189     191
cleanup        275     307     312     344
cnt            378     392     404     441     443    1242    1244
cp            1241    1246
crc            534     613     729+    749+    805     808     811     833+    988    1159    1179+
      1191    1240    1246+   1249
crctab        1221    1234    1246
csp_fgets      365    1045
ctype          149
cur            161     332     335     337     338     888     890    1011+   1015    1017    1021
      1022    1025
curpart        520     613     774     777     782     814     982    1087    1088    1131    1148
      1191    1199
defined        496     902     912
disp_help      183     195     282     480
done           906     919     921     934     951
e      372     409     414     446     451+    454     458     461
ext            977     993     994     996    1088    1131
extr_file      272     510
extract        170     199     262
fclose         348     349     350     355     587     594     628     742    1049    1113    1168
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * CSplit.h
 * 08/31/93  Fred Cole     original
 */

#ifndef  __STDC__
 #include "dirport.h"
#endif

#define VERSION   "2.2"

#ifndef B_FALSE
#define B_FALSE 0
#endif

#ifndef B_TRUE
#define B_TRUE  1
#endif

enum { NOERR = 0, SYNTAX, ABORT, MEMORY, FILEIO, PROCESS };

#define FNAMELEN  12    /* 8.3 filename character count     */
#define LLENMAX  128    /* maximum input source line length */
#define PATHMAX   80    /* maximum path length              */
#define MAXFSPEC (PATHMAX + FNAMELEN)

#define LENMIN    25    /* minimum source lines per section */
#define LENDEF    90    /* default source lines per section */ /* TS */
#define LENMAX SHRT_MAX /* maximum source lines per section */
#define TABMIN     2    /* minimum spaces per tab char      */
#define TABDEF     4    /* default spaces per tab char      */
#define TABMAX    16    /* maximum spaces per tab char      */
#define WIDMIN    55    /* minimum source chars per line    */
#define WIDDEF    75    /* default source chars per line    */
#define WIDMAX   255    /* maximum source chars per line    */

#define SEP_CLEN 10     /* separator character string length */
#define SEP_CDEF '>'    /* default separator character */
#define SEP_BF   "Begin file "
#define SEP_BP   "Begin part "
#define SEP_EF   "End file "
#define SEP_EP   "End part "
#define SEP_ID   "CSplit: "
#define SEP_VR   "Version "

typedef struct _slst
{
  char srcfile[MAXFSPEC + 1];     /* path and filename */
  struct _slst *next;
} SLST;

SLST *add_list  (char *);
void  cleanup   (void);
char *csp_fgets (char *, int, FILE *, int);
void  disp_help (void);
int   extr_file (char *, char);
void  free_list (void);
int   init_list (int, char **, int);
int   split_src (SLST *, char *, int, int, int, char);
unsigned short updcrc (unsigned short, unsigned char *, unsigned int);
void  initcrctab (void);
```

```
TEXT STATISTICS

  1956 characters
    63 lines

LEXICAL STATISTICS

    18 comments [std-C]
     0 comments [C++]
    31 preprocessor instructions
     1 constants [character]
     8 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    ABORT           22
    B_FALSE         14      15
    B_TRUE          18      19
    FILE            56
    FILEIO          22
    FNAMELEN        24      27
    LENDEF          30
    LENMAX          31
    LENMIN          29
    LLENMAX         25
    MAXFSPEC        27      50
    MEMORY          22
    NOERR           22
    PATHMAX         26      27
    PROCESS         22
    SEP_BF          41
    SEP_BP          42
    SEP_CDEF        40
    SEP_CLEN        39
    SEP_EF          43
    SEP_EP          44
    SEP_ID          45
    SEP_VR          46
    SHRT_MAX        31
    SLST            52      54      61
    SYNTAX          22
    TABDEF          33
    TABMAX          34
    TABMIN          32
    VERSION         12
    WIDDEF          36
    WIDMAX          37
    WIDMIN          35
    __STDC__         8
    _slst           48      51
    add_list        54
    cleanup         55
    csp_fgets       56
    disp_help       57
    extr_file       58
    free_list       59
    init_list       60
    initcrctab              63
    next            51
    split_src       61
    srcfile         50
    updcrc          62
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Print a line of text, displaying Ctrl characters using leading carets
  5 | **  public domain by Bob Stout
  6 | */
  7 |
  8 | #include <stdio.h>
  9 | #include <string.h>
 10 | #include "ctrlprnt.h"
 11 |
 12 | void ctrl_print(char *line)
 13 | {
 14 |       while (*line)
 15 |       {
 16 |             if (' ' > *line)
 17 |             {
 18 |                   putchar('^');
 19 |                   putchar('@' + (*line++));
 20 |             }
 21 |             else  putchar(*line++);
 22 |       }
 23 |       if (!strcmp((line - 2), "\x0d\x0a") || !strcmp((line - 2), "\x0a\x0d"))
 24 |             putchar('\n');
 25 | }
 26 |
 27 | #ifdef TEST
 28 |
 29 | #include <stdlib.h>
 30 | #include <ctype.h>
 31 |
 32 | main()
 33 | {
 34 |       char *p, *test = "This is a test";
 35 |
 36 |       for (p = strupr(test); *p; ++p)
 37 |       {
 38 |             if (isalpha(*p))
 39 |                   *p = *p - 64;
 40 |       }
 41 |       ctrl_print(test);
 42 |       return EXIT_SUCCESS;
 43 | }
 44 |
 45 | #endif /* TEST */
```

TEXT STATISTICS

```
   895 characters
    45 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     4 constants [character]
     4 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS            42
    TEST            27
    ctrl_print              12      41
    ctype           30
    h       8        9      29      30
    isalpha         38
    line            12      14      16      19      21      23+
    main            32
    p       34      36+     38      39+
    putchar         18      19      21      24
    stdio            8
    stdlib          29
    strcmp          23+
    string           9
    strupr          36
    test            34      36      41
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header file for CTRLPRNT.C
   5 | */
   6 |
   7 | #ifndef CTRLPRNT__H
   8 | #define CTRLPRNT__H
   9 |
  10 | void ctrl_print(char *line);
  11 |
  12 | #endif /* CTRLPRNT__H */


TEXT STATISTICS

   197 characters
    12 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    CTRLPRNT__H              7       8
    ctrl_print              10
    line            10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  CUBIC.C - Solve a cubic polynomial
**  public domain by Ross Cottrell
*/

#include <math.h>
#include <stdlib.h>
#include "snipmath.h"

void SolveCubic(double  a,
                double  b,
                double  c,
                double  d,
                int    *solutions,
                double *x)
{
      long double    a1 = b/a, a2 = c/a, a3 = d/a;
      long double    Q = (a1*a1 - 3.0*a2)/9.0;
      long double R = (2.0*a1*a1*a1 - 9.0*a1*a2 + 27.0*a3)/54.0;
      double    R2_Q3 = R*R - Q*Q*Q;

      double    theta;

      if (R2_Q3 <= 0)
      {
            *solutions = 3;
            theta = acos(R/sqrt(Q*Q*Q));
            x[0] = -2.0*sqrt(Q)*cos(theta/3.0) - a1/3.0;
            x[1] = -2.0*sqrt(Q)*cos((theta+2.0*PI)/3.0) - a1/3.0;
            x[2] = -2.0*sqrt(Q)*cos((theta+4.0*PI)/3.0) - a1/3.0;
      }
      else
      {
            *solutions = 1;
            x[0] = pow(sqrt(R2_Q3)+fabs(R), 1/3.0);
            x[0] += Q/x[0];
            x[0] *= (R < 0.0) ? 1 : -1;
            x[0] -= a1/3.0;
      }
}

#ifdef TEST

int main(void)
{
      double  a1 = 1.0, b1 = -10.5, c1 = 32.0, d1 = -30.0;
      double  a2 = 1.0, b2 = -4.5, c2 = 17.0, d2 = -30.0;
      double  x[3];
      int     solutions;

      SolveCubic(a1, b1, c1, d1, &solutions, x);

      /* should get 3 solutions: 2, 6 & 2.5   */

      SolveCubic(a2, b2, c2, d2, &solutions, x);

      /* should get 1 solution: 2.5           */

      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1577 characters
    64 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
    44 constants [numeric]
```

SYMBOL TABLE

PI		31	32								
Q	20	22+	29+	30	31	32	38				
R	21	22+	29	37	39						
R2_Q3		22	26	37							
SolveCubic			12	53	57						
TEST		44									
a	12	19+									
a1		19	20+	21+	30	31	32	40	48	53	
a2		19	20	21	49	57					
a3		19	21								
acos		29									
b	13	19									
b1		48	53								
b2		49	57								
c	14	19									
c1		48	53								
c2		49	57								
cos		30	31	32							
d	15	19									
d1		48	53								
d2		49	57								
fabs		37									
h	8	9									
main		46									
math		8									
pow		37									
solutions		16	28	36	51	53	57				
sqrt		29	30	31	32	37					
stdlib		9									
theta		24	29	30	31	32					
x	17	30	31	32	37	38+	39	40	50	53	57

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  Program to set the size of the cursor
 5 | **
 6 | **  Public domain demonstration by Bob Jarvis
 7 | */
 8 |
 9 | #include <stdio.h>      /* puts()               */
10 | #include <dos.h>        /* int86(), union REGS  */
11 | #include <stdlib.h>     /* exit(), atoi()       */
12 |
13 | char *help =  "CURSIZE - sets the cursor size.\n"
14 |               "Usage:\n"
15 |               "   CURSIZE <top-line> <bottom-line>\n"
16 |               "where\n"
17 |               "   top-line = top line of cursor within character cell\n"
18 |               "   bottom-line = bottom line\n"
19 |               "Example:\n"
20 |               "   CURSIZE 7 8     <set cursor to bottom 2 lines of VGA>\n"
21 |               "   CURSIZE 32 32   <turns cursor off>";
22 |
23 | void cursor_size(int top_line, int bottom_line)
24 | {
25 |       union REGS regs;
26 |
27 |       regs.h.ah = 1;
28 |       regs.h.ch = (unsigned char)top_line;
29 |       regs.h.cl = (unsigned char)bottom_line;
30 |
31 |       int86(0x10,&regs,&regs);
32 | }
33 |
34 | void get_cursor_size(int *top_line, int *bottom_line)
35 | {
36 |       union REGS regs;
37 |
38 |       regs.h.ah = 3;
39 |       regs.h.bh = 0;
40 |       int86(0x10, &regs, &regs);
41 |
42 |       *top_line = regs.h.ch;
43 |       *bottom_line = regs.h.cl;
44 |
45 |       return;
46 | }
47 |
48 | main(int argc, char *argv[])
49 | {
50 |       int top, bottom;
51 |
52 |       if(argc < 3)
53 |       {
54 |             puts(help);
55 |             exit(1);
56 |       }
57 |
58 |       top = atoi(argv[1]);
59 |       bottom = atoi(argv[2]);
60 |
61 |       cursor_size(top,bottom);
62 |
63 |       top = bottom = -1;
64 |
65 |       get_cursor_size(&top, &bottom);
66 |
67 |       printf("top = %d  bottom = %d\n", top, bottom);
68 |       return 0;
69 | }
```

TEXT STATISTICS

```
  1611 characters
    69 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    10 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    REGS            25      36
    ah              27      38
    argc            48      52
    argv            48      58      59
    atoi            58      59
    bh              39
    bottom          50      59      61      63      65      67
    bottom_line             23      29      34      43
    ch              28      42
    cl              29      43
    cursor_size             23      61
    dos             10
    exit            55
    get_cursor_size         34      65
    h       9       10      11      27      28      29      38      39      42      43
    help            13      54
    int86           31      40
    main            48
    printf          67
    puts            54
    regs            25      27      28      29      31+     36      38      39      40+     42      43
    stdio            9
    stdlib          11
    top             50      58      61      63      65      67
    top_line        23      28      34      42
```

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************/
/**   CURSOR()                                              **/
/**   ARGUMENTS: A char variable identifying what to do with    **/
/**              the cursor.                                **/
/**   RETURN: none                                          **/
/**                                                         **/
/**   DESCRIPTION:  This function receives a character which    **/
/**                 tells it to do one of several things.       **/
/**                 Turn the cursor on or off, change its size, **/
/**                 save the cursor size & position, restore the**/
/**                 size, restore the position, or restore both.**/
/**                                                         **/
/**   COMMANDS:   'S' - Save cursor size and position           **/
/**               'R' - Restore cursor size and position        **/
/**               'P' - Restore cursor position                 **/
/**               'Z' - Restore cursor size                     **/
/**               'H' - Hide cursor (cursor off)                **/
/**               'O' - Cursor on                               **/
/**               'N' - Cursor normal (same  as 'O')            **/
/**               'B' - Cursor big                              **/
/**                                                         **/
/**   BY Bill Wilkie, 1988; Bob Stout, 1995                 **/
/*************************************************************/

#include <dos.h>
#include <ctype.h>
#include "cursor.h"

static unsigned position;       /* global to hold cursor position*/
static unsigned size;           /* global to hold cursor size    */

void cursor(int tmp)
{
      union REGS inregs,outregs;               /* cpu registers */

      switch (toupper(tmp))
      {
      case 'S' :                              /* SAVE CURSOR DATA */
            inregs.h.ah = 3;    /* read cursor position and size */
            inregs.h.bh = 0;                   /* from page zero */
            int86(0x10,&inregs,&outregs);
            position = outregs.x.dx;           /* store position */
            size = outregs.x.cx;                   /* store size */
            break;

      case 'P' :                       /* RESTORE CURSOR POSITION */
            inregs.h.ah = 2;              /* set cursor position */
            inregs.h.bh = 0;                     /* on page zero */
            inregs.x.dx = position;      /* at this old position */
            int86(0x10,&inregs,&outregs);
            break;

      case 'R' :                /* RESTORE CURSOR SIZE & POSITION */
            inregs.h.ah = 2;              /* set cursor position */
            inregs.h.bh = 0;                     /* on page zero */
            inregs.x.dx = position;      /* at this old position */
            int86(0x10,&inregs,&outregs);
            /* fall through to... */

      case 'Z' :                             /* RESTORE CURSOR SIZE */
            inregs.h.ah = 1;                  /* set cursor size */
            inregs.x.cx = size;                /* to saved value */
            int86(0x10,&inregs,&outregs);
            break;

      case 'H' :                                     /* CURSOR OFF */
            inregs.h.ah = 1;                    /* set cursor size */
            inregs.h.ch = 0x20;      /* set bit turns cursor off */
            int86(0x10,&inregs,&outregs);
            break;

      case 'O' :                                      /* CURSOR ON */
      case 'N' :                                   /* CURSOR NORMAL */
            inregs.h.ah = 1;                    /* set cursor size */
            inregs.h.ch = 6;                  /* cursor start line */
            inregs.h.cl = 7;                    /* cursor end line */
            int86(0x10,&inregs,&outregs);
            break;

      case 'B' :                                     /* CURSOR BIG */
            inregs.h.ah = 1;                    /* set cursor size */
            inregs.h.ch = 4;                  /* cursor start line */
            inregs.h.cl = 7;                    /* cursor end line */
            int86(0x10,&inregs,&outregs);
            break;
      }
}
```

TEXT STATISTICS

```
  4383 characters
    89 lines
```

LEXICAL STATISTICS

```
    56 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     8 constants [character]
     1 constants [string]
    22 constants [numeric]
```

SYMBOL TABLE

```
    REGS            36
    ah              41      49      56      63      69      76      83
    bh              42      50      57
    ch              70      77      84
    cl              78      85
    ctype           28
    cursor          34
    cx              45      64
    dos             27
    dx              44      51      58
    h       27      28      41      42      49      50      56      57      63      69      70      76
            77      78      83      84      85
    inregs          36      41      42      43      49      50      51      52      56      57      58
            59      63      64      65      69      70      71      76      77      78      79      83
            84      85      86
    int86           43      52      59      65      71      79      86
    outregs         36      43      44      45      52      59      65      71      79      86
    position        31      44      51      58
    size            32      45      64
    tmp             34      38
    toupper         38
    x       44      45      51      58      64
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  Header file for CURSOR.C
   5 | */
   6 |
   7 | #ifndef CURSOR__H
   8 | #define CURSOR__H
   9 |
  10 | void cursor(int tmp);
  11 |
  12 | #endif /* CURSOR__H */


TEXT STATISTICS

   173 characters
    12 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    CURSOR__H        7       8
    cursor          10
    tmp             10
```

```
/* +++Date last modified: 05-Jul-1997 */

/* Extract comments from a C program source file.
**
** This program acts as a filter to copy comments in a C source
** file to the output.  Each comment includes the starting and
** ending delimiters and is followed by a newline.
**
** Three #ifdef options are defined:
**   INHIBIT_TRIGRAPHS prevents recognition of trigraphs, which
**      can affect detection of escaped characters,
**      i.e., ??/" is an escaped quote.
**   TRANSLATE_TRIGRAPHS causes the output to have trigraphs
**      converted to the normal C characters.
**   CPP_MODE causes "//" to start a comment.
** The default for these symbols is undefined, resulting in
** operation on strict ANSI source, except as noted below.
**
** What makes this program interesting is that comment detection
** should be inhibited within strings and character constants.
**
** Note: The name of a header following #include can, under ANSI,
** contain any sequence of characters, except \n and the closing
** > or ".  This program doesn't inhibit comment, string, or character
** constant detection within the header name, as an ANSI parser must.
**
** Written by and contributed to the public domain by
** Thad Smith III,   Boulder, CO,  October 1990.
*/

#include <stdio.h>

#ifndef INHIBIT_TRIGRAPHS        /* default: recognize trigraphs */
 #define getnc()  getnsc(1)       /* get char with trigraph xlate */
 #ifdef TRANSLATE_TRIGRAPHS
  #define getcmtc() getnsc(1)     /* get comment char w/ t.g. xlate */
 #else
  #define getcmtc() getnsc(0)     /* default: no comment t.g. xlate */
 #endif

/*
**  get next source character or EOF
*/

int getnsc(int cvtg)             /* boolean: convert trigraphs */
{
      static int c, nc, nnc;     /* next 3 characters */

      /* shift in next source character */

      c = nc;  nc = nnc;   nnc = getchar();

      /* perform trigraph substitution */

      if (cvtg && c == '?' && nc == '?')
      {
            switch (nnc)
            {
            case '=' :
                  c = '#' ;
                  break;
            case '(' :
                  c = '[' ;
                  break;
            case '/' :
                  c = '\\';
                  break;
            case ')' :
                  c = ']' ;
                  break;
            case '\'':
                  c = '^' ;
                  break;
            case '<' :
                  c = '{' ;
                  break;
            case '!' :
                  c = '|' ;
                  break;
            case '>' :
                  c = '}' ;
                  break;
            case '-' :
                  c = '~' ;
                  break;
            default  :
                  return c;        /* no substitution */
            }
            nc = getchar();   nnc = getchar();
      }
      return c;
}

#else       /* don't process trigraphs */

 #define getnc()   getchar()
 #define getcmtc() getchar()
#endif

int main(void)
```

```
{
      int pc;  /* previous character      */
      int c;   /* current input character */

#ifndef INHIBIT_TRIGRAPHS
      getnc();                                    /* prime the pump */
      getnc();
#endif
      c = getnc();                                /* get first char */

      for (;;)                                 /* in non-comment area */
      {
            switch (c)
            {
            case '/':                       /* possible start of comment */
                  if ((c= getnc()) == '*')      /* process comment */
                  {
                        putchar('/');
                        putchar('*');

                        /* copy comment to stdout */

                        for (pc = 0; (c = getcmtc()) != EOF &&
                              (putchar(c) != '/' || pc != '*'); pc=c)
                                    ;
                              putchar('\n');
#ifdef CPP_MODE
                  }
                  else if (c == '/')      /*  '//' comment */
                  {
                        putchar('/');
                        putchar('/');
                        while ((c = getcmtc()) != EOF && putchar(c) != '\n')
                              ;
#endif
                  }
                  else  continue;         /* test current char */
                  break;

            case '\"':                                  /* start of string */
            case '\'':    /* start of (possibly multi-byte) char constant */
                  pc = c;                               /* save delimiter */
                  do                            /* scan through character constant,
                                                ** discarding escape chars
                                                */
                  {
                        while ((c = getnc()) == '\\')
                              getnc();
                  } while (c != pc && c != EOF);
                  break;
            }
            if (c == EOF)
                  return 0;
            else  c = getnc();
      }
}
```

```
TEXT STATISTICS

  5182 characters
   156 lines

LEXICAL STATISTICS

    27 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
    34 constants [character]
     0 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    CPP_MODE       127
    EOF            123     133     149     152
    INHIBIT_TRIGRAPHS       33     105
    TRANSLATE_TRIGRAPHS             35
    c       47      51      55      60      63      66      69      72      75      78      81      84
            87      91     103     109     113     116     123     124+    129     133+    142     147
           149+    152     154
    cvtg            45      55
    getchar         51      89+     96      97
    getcmtc         36      38      97     123     133
    getnc           34      96     106     107     109     116     147     148     154
    getnsc          34      36      38      45
    h       31
    main           100
    nc              47      51+     55      89
    nnc             47      51+     57      89
    pc             102     123     124+    142     149
    putchar        118     119     124     126     131     132     133
    stdio           31
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Header for SNIPPETS date & time function, excluding scalar dates
*/

#ifndef DATETIME__H
#define DATETIME__H

#include "sniptype.h"

/*
**  Julian day number functions from JDN_L.C
*/

long ymd_to_jdnl(int y, int m, int d, int julian);
void jdnl_to_ymd(long jdn, int *yy, int *mm, int *dd, int julian);

/*
**  Add times from ADDTIME.C
*/

int add_time(unsigned basehrs, unsigned basemins, unsigned basesecs,
             int spanhrs, int spanmins, int spansecs,
             unsigned *hrs, unsigned *mins, unsigned *secs, int *days);

/*
**  Determine the date of Easter for any given year
*/

void easter(int year, int *easter_month, int *easter_day);


/*
**  Moon phase from MOON_AGE.C
*/

int moon_age(int month, int day, int year);

/*
**  Date parser from PARSDATE.C
*/

typedef enum {USA, ISO, EUROPE} Syntax_T;

Boolean_T parse_date(const char *str, unsigned *year, unsigned *month,
                     unsigned *day, Syntax_T syntax);

/*
**  Time parser from PARSTIME.C
*/

Boolean_T parse_time(const char *str, unsigned *hours, unsigned *mins,
                     unsigned *secs);


#endif /* DATETIME__H */
```

TEXT STATISTICS

```
  1228 characters
    57 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       46      53
    DATETIME__H              7       8
    EUROPE          44
    ISO             44
    Syntax_T        44      47
    USA             44
    add_time        23
    basehrs         23
    basemins        23
    basesecs        23
    d       16
    day             38      47
    days            25
    dd              17
    easter          31
    easter_day              31
    easter_month            31
    hours           53
    hrs             25
    jdn             17
    jdnl_to_ymd             17
    julian          16      17
    m       16
    mins            25      53
    mm              17
    month           38      46
    moon_age        38
    parse_date              46
    parse_time              53
    secs            25      54
    spanhrs         24
    spanmins        24
    spansecs        24
    str             46      53
    syntax          47
    y       16
    year            31      38      46
    ymd_to_jdnl             16
    yy              17
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
*@(#) date_mac_conv() - Converting a date string with the format from
*@(#)    macro __DATE__ to ISO standard.
*
*****************************************************************************
*@(#)1992 Erik Bachmann  (E-mail: ebp@dde.dk)
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <stdio.h>                    /* scanf, sprintf */
#include <string.h>                   /* strcpy, strcmp */
#include "bacstd.h"                   /* _CfnTYPE */

/*
 /-------------------------------------\
|     DATE_MAC_CONV                    |-----------------------------------|
|\-------------------------------------/
| Converting a date string with the format from __DATE__ to ISO standard
|
|
|
| Example:   "Sep 16 1992" -> "1992-09-16"
|
|-----------------------------------------------------------------------------|
| CALL:
|    strcpy(string, date_mac_conv(string) ) ;
|
| HEADER:
|    stdio.h     : scanf, sprint
|    string.h    : strcpy, strcmp
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszDate     : String containing __DATE__ format.
|
| PROTOTYPE:
|    char _CfnTYPE *date_mac_conv(char *pszDate) ;
|
| RETURN VALUE:
|    char        : ISO date format YYYY-MM-DD (length 8 char)
|
| MODULE:
|    date__.c
|-----------------------------------------------------------------------------|
|
|
|
|
|-----------------------------------------------------------------------------|
|1992-09-16/Erik Bachmann
\-----------------------------------------------------------------------------|*/

char _CfnTYPE *date_mac_conv(char *pszDate)
{
   char  *paMonth[12] = {"Jan", "Feb", "Mar", "Apr","May", "Jun", "Jul", "Aug",
                         "Sep", "Oct", "Nov", "Dec"} ;
                                      /* Names of months */

   char  szString[12] ;               /* String fore convertion */
   char  cMonth[4],                   /* Month */
         cDay[3],                     /* Day */
         cYear[5] ;                   /* Year */
   int   iCounter  ;                  /* Local counter */

   /*-------------------------------------------------------------*/

   strcpy(szString, pszDate) ;           /* Get string to convert */

   sscanf(szString, "%s %s %s", cMonth, cDay, cYear) ;
                                      /* Split string */

   for ( iCounter = 11  ; iCounter >= 0 ; iCounter--)
                                      /* loop for testing month */
   {
      if (strcmp(cMonth, paMonth[iCounter]) == 0)
                                      /* - IF month is found */
      {
         sprintf(cMonth, "%0.2i", iCounter + 1 ) ;
                                      /* - - Insert counter + 1 as month */
         break ;
      }
   }

   /* Check valid day */
   iCounter = atoi( cDay ) ;
   if ( ( 1 > iCounter ) || ( 31 < iCounter ) )
      strcpy( cDay, "01" ) ;

   sprintf(szString, "%04.4s-%02.2s-%02.2s", cYear,cMonth,cDay) ;
                                      /* Merge ISO date */

   return( (char _CfnTYPE *) szString ) ;
}  /*** date_mac_conv() ***/

#ifdef TEST
```

```
 101 | main()
 102 | {
 103 |    char  szString[20] ;
 104 |
 105 |    /*--------------------------------*/
 106 |
 107 |    printf( "\n%s\n", date_mac_conv( __DATE__ ) ) ;
 108 |
 109 |    return( 0 ) ;
 110 | }
 111 | #endif
```

TEXT STATISTICS

```
  5463 characters
   111 lines
```

LEXICAL STATISTICS

```
    22 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
    18 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    TEST           100
    _CfnTYPE        58      97
    __DATE__       107
    atoi            90
    cDay            66      74      90      92      94
    cMonth          65      74      80      83      94
    cYear           67      74      94
    date_mac_conv           58     107
    h       13      14
    iCounter        68      77+     80      83      90      91+
    main           101
    paMonth         60      80
    printf         107
    pszDate         58      72
    sprintf         83      94
    sscanf          74
    stdio           13
    strcmp          80
    strcpy          72      92
    string          14
    szString        64      72      74      94      97     103
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DAYNUM.C - Functions to return statistics about a given date.
**
**  public domain by Bob Stout - uses Ray Gardner's SCALDATE.C
*/

#include "scaldate.h"

static long jan1date;

/*
**  Determine if a given date is valid
*/

Boolean_T valiDate(unsigned yr, unsigned mo, unsigned day)
{
      unsigned int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

      if (1 > mo || 12 < mo)
            return False_;
      if (1 > day || day > (days[mo - 1] + (2 == mo && isleap(yr))))
            return False_;
      else  return True_;
}

/*
**  Return the day of the week
*/

unsigned dow(unsigned yr, unsigned mo, unsigned day)
{

#if (!ISO_CAL)    /* Sunday(0) -> Saturday(6) (i.e. U.S.) calendars  */
      return (unsigned)(ymd_to_scalar(yr, mo, day) % 7L);
#else             /* International Monday(0) -> Sunday(6) calendars  */
      return (unsigned)((ymd_to_scalar(yr, mo, day) - 1L) % 7L);
#endif
}

/*
**  Return the day of the year (1 - 365/6)
*/

int daynum(int year, int month, int day)
{
      jan1date = ymd_to_scalar(year, 1, 1);
      return (int)(ymd_to_scalar(year, month, day) - jan1date + 1L);
}

/*
**  Return the week of the year (1 - 52, 0 - 52 if ISO)
*/

int weeknum(int year, int month, int day)
{
      int wn, j1n, dn = daynum(year, month, day);

      dn += (j1n = (int)((jan1date - (long)ISO_CAL) % 7L)) - 1;
      wn = dn / 7;
      if (ISO_CAL)
            wn += (j1n < 4);
      else  ++wn;
      return wn;
}

/*
**  Return the phase of the moon (0 -> 7, 0 = new, 4 = full)
*/

char *MoonPhaseText[8] = {"new", "waxing crescent", "first quarter",
      "waxing gibbous", "full", "waning gibbous", "third quarter",
      "waning crescent"
      };

unsigned moonphase(unsigned yr, unsigned mo, unsigned dy)
{
      unsigned long date = (unsigned long)ymd_to_scalar(yr, mo, dy);

      date *= 9693L;
      date /= 35780L;
      date -= 4L;
      date %= 8L;
      return (unsigned)date;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

void do_err(void);

main(int argc, char *argv[])
{
      int day, month, year;
      char *days[] =
#if (!ISO_CAL)
            {"Sunday", "Monday", "Tuesday", "Wednesday",
```

```
101 |                "Thursday", "Friday", "Saturday"};
102 | #else
103 |               {"Monday", "Tuesday", "Wednesday", "Thursday",
104 |                "Friday", "Saturday", "Sunday"};
105 | #endif
106 |
107 |       if (4 > argc)
108 |       {
109 |             puts("Usage: DAYNUM month day year");
110 |             return EXIT_FAILURE;
111 |       }
112 |
113 |       month = atoi(argv[1]);
114 |       day   = atoi(argv[2]);
115 |       year  = atoi(argv[3]);
116 |       if (100 > year)
117 |             year += 1900;
118 |
119 |       if (!valiDate(year, month, day))
120 |             printf("%d/%d/%d is invalid!\n", month, day, year);
121 |       else
122 |       {
123 |             printf("%d/%d/%d is a %s, day #%d in week %d\n", month, day, year,
124 |                   days[dow(year, month, day)], daynum(year, month, day),
125 |                   weeknum(year, month, day));
126 |             printf("The phase of the moon is %s\n",
127 |                   MoonPhaseText[moonphase(year, month, day)]);
128 |       }
129 |       return EXIT_SUCCESS;
130 | }
131 |
132 | #endif /* TEST */
```

TEXT STATISTICS

```
  3252 characters
   132 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
    27 constants [string]
    38 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       17
    EXIT_FAILURE           110
    EXIT_SUCCESS           129
    False_          22      24
    ISO_CAL         35      60      62      99
    MoonPhaseText           72     127
    TEST            88
    True_           25
    argc            95     107
    argv            95     113     114     115
    atoi           113     114     115
    date            79      81      82      83      84      85
    day             17      23+     32      36      38      46      49      56      58      97     114
          119     120     123     124+    125     127
    daynum          46      58     124
    days            19      23      98     124
    dn              58      60      61
    do_err          93
    dow             32     124
    dy              77      79
    h       90      91
    isleap          23
    j1n             58      60      63
    jan1date        11      48      49      60
    main            95
    mo              17      21+     23+     32      36      38      77      79
    month           46      49      56      58      97     113     119     120     123     124+    125
          127
    moonphase       77     127
    printf         120     123     126
    puts           109
    stdio           90
    stdlib          91
    valiDate        17     119
    weeknum         56     125
    wn              58      61      63      64      65
    year            46      48      49      56      58      97     115     116     117     119     120
          123     124+    125     127
    ymd_to_scalar           36      38      48      49      79
    yr              17      23      32      36      38      77      79
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DBL2LONG.C - Functions to round doubles to longs
**  Public domain by Ross G. Cottrell, June 1992
*/

#include <float.h>
#include <limits.h>
#include "snipmath.h"

#if defined(__ZTC__)
 #include <fltenv.h>
 #define SAVEROUND() \
       int fersave = fegetround(); \
       fesetround(FE_TONEAREST)
 #define RESTOREROUND() \
       fesetround(fersave)
#else
 #if !defined(FLT_ROUNDS) || (FLT_ROUNDS != 1)
  #error FLT_ROUNDS needs to be defined to be 1
 #endif
 #define SAVEROUND()
 #define RESTOREROUND()
#endif

/* Assume IEEE doubles, little-endian CPU, 32-bit 2's complement longs. */
/* (Actually, the assumptions made here aren't quite that gross.)       */

unsigned long dbl2ulong(double t)
{
      SAVEROUND();
      t += 1.0 / DBL_EPSILON;
      RESTOREROUND();
      return *(unsigned long *)&t;
}

long dbl2long(double t)
{
      SAVEROUND();
      t += 1.0 / DBL_EPSILON + 2.0 * (LONG_MAX + 1.0);
      RESTOREROUND();
      return *(long *)&t;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif
#ifdef __TURBOC__
 #pragma argsused
#endif

int main(int argc, char **argv)
{
      while (*++argv)
      {
            printf("'%s', as a long: %ld, as an unsigned long: %lu\n",
                  *argv, dbl2long(atof(*argv)), dbl2ulong(atof(*argv)));
      }
      return 0;
}

#endif /* TEST */

/*

EXPLANATION:

The offset of 1.0/DBL_EPSILON forces the least significant bit of the
mantissa to represent the integer 1.  This may not work on all formats of
doubles, but I think it's a safe bet for IEEE compliant doubles, and any
other floating point format with a radix of 2.  When this offset is added,
the number should be rounded to the nearest representable value.  If the
compiler does not support extended math funtionality, a compile-time error is
generated if FLT_ROUNDS isn't set to 1, i.e. if not rounding to the nearest
representable value.  The addition of 2.0*(LONG_MAX+1.0) for the signed long
is to prevent the MSB of the mantissa being borrowed for negative inputs - if
this happened, the exponent would change and the LSB of the mantissa would no
longer be worth 1.  This offset would be perfectly okay to use with the
unsigned longs too but it's unnecessary for them, unless you want to get the
answer correct modulo 2^^32 for negatives.

*/
```

TEXT STATISTICS

```
  2488 characters
    89 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    25 preprocessor instructions
     0 constants [character]
     2 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    DBL_EPSILON             33      41
    FE_TONEAREST            16
    FLT_ROUNDS              20+     21
    LONG_MAX        41
    RESTOREROUND            17      24      34      42
    SAVEROUND       14      23      32      40
    TEST            46
    __TURBOC__              55
    __WATCOMC__             52
    __ZTC__         12
    argc            59
    argsused        56
    argv            59      61      64+
    atof            64+
    be              21+
    dbl2long        38      64
    dbl2ulong       30      64
    defined         12      20      21
    fegetround              15
    fersave         15      18
    fesetround              16      18
    fltenv          13
    h       8        9      13      48      49      50
    limits           9
    main            59
    math            50
    needs           21
    off             53
    printf          63
    stdio           48
    stdlib          49
    t      30       33      35      38      41      43
    to              21+
    unreferenced            53
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DBLROUND.C - Rounds a double to the nearest whole number
**  public domain by Ross Cottrell
*/

#include <float.h>
#include <limits.h>
#include "snipmath.h"

#if defined(__ZTC__)
 #include <fltenv.h>
 #define SAVEROUND() \
        int fersave = fegetround(); \
        fesetround(FE_TONEAREST)
 #define RESTOREROUND() \
        fesetround(fersave)
#else
 #if !defined(FLT_ROUNDS) || (FLT_ROUNDS != 1)
  #error FLT_ROUNDS needs to be defined to be 1
 #endif
 #define SAVEROUND()
 #define RESTOREROUND()
#endif

double dround(double x)
{
      Boolean_T flag;
      static volatile double X;

      SAVEROUND();
      if (True_ == (flag = (x < 0.0)))
            X = -x;
      else  X = x;
      X += 1.0 / DBL_EPSILON;
      X -= 1.0 / DBL_EPSILON;
      RESTOREROUND();
      return ((flag) ? -X : X);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      double val;
      char *dummy;

      while (--argc)
      {
            val = strtod((const char *)(*(++argv)), &dummy);
            printf("dround(%g) = ", val);
            printf("%.12g\n", dround(val));
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1266 characters
    61 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    18 preprocessor instructions
     0 constants [character]
     3 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       29
    DBL_EPSILON             36      37
    EXIT_SUCCESS            58
    FE_TONEAREST            16
    FLT_ROUNDS              20+     21
    RESTOREROUND            17      24      38
    SAVEROUND       14      23      32
    TEST            42
    True_           33
    X       30      34      35      36      37      39+
    __ZTC__         12
    argc            47      52
    argv            47      54
    be              21+
    defined         12      20      21
    dround          27      56
    dummy           50      54
    fegetround              15
    fersave         15      18
    fesetround              16      18
    flag            29      33      39
    fltenv          13
    h        8       9      13      44      45
    limits           9
    main            47
    needs           21
    printf          55      56
    stdio           44
    stdlib          45
    strtod          54
    to              21+
    val             49      54      55      56
    x       27      33      34      35
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DELAY.C - A portable time delay compatible with Borland's and Watcom's
**            delay() function.
**
**  public domain demo by Bob Stout
*/

#if (!defined(__WATCOMC__) && !defined(__TURBOC__)) || (defined(__TURBOC__) \
      && (defined(_Windows) && !defined(__DPMI16) && !defined(__DPMI32__)))

#include <time.h>
#include "delay.h"

#ifndef  CLOCKS_PER_SEC                   /* CLOCKS_PER_SEC is ANSI/ISO */
 #define CLOCKS_PER_SEC CLK_TCK
#endif

void delay(unsigned short msec)
{
      clock_t t0;
      unsigned long diff = 0L;

      for (t0 = clock(); diff < (unsigned long)msec; )
      {
            diff  = (unsigned long)(clock() - t0);
            diff *= 1000L;
            diff /= CLOCKS_PER_SEC;
      }
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      int msec;

      if (2 > argc)
      {
            puts("Usage: DELAY milliseconds");
            return EXIT_FAILURE;
      }
      msec = atoi(argv[1]);
      printf("Delaying %d milliseconds\n", msec);
      delay(msec);
      return EXIT_SUCCESS;
}

#endif /* TEST */

#endif /* Not Watcom or Borland */
```

TEXT STATISTICS

```
  1228 characters
    55 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     3 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    CLK_TCK         17
    CLOCKS_PER_SEC          16      17      29
    EXIT_FAILURE            45
    EXIT_SUCCESS            50
    TEST            33
    _Windows        11
    __DPMI16        11
    __DPMI32__              11
    __TURBOC__              10+
    __WATCOMC__             10
    argc            38      42
    argv            38      47
    atoi            47
    clock           25      27
    clock_t         22
    defined         10+     11+
    delay           20      49
    diff            23      25      27      28      29
    h       13      35      36
    main            38
    msec            20      25      40      47      48      49
    printf          48
    puts            44
    stdio           35
    stdlib          36
    t0              22      25      27
    time            13
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DELAY.H - SNIPPETS header file for DELAY.C
  5 | */
  6 |
  7 | #ifndef DELAY__H
  8 | #define DELAY__H
  9 |
 10 | #if defined(__WATCOMC__) || (defined(__TURBOC__) && \
 11 |       (!defined(_Windows)) || defined(__DPMI16__) || defined(__DPMI32__))
 12 |  #include <dos.h>
 13 | #else
 14 |  void delay(unsigned short msec);
 15 | #endif
 16 |
 17 | #endif /* DELAY__H */
```

TEXT STATISTICS

```
   364 characters
    17 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DELAY__H          7       8
    _Windows         11
    __DPMI16__                11
    __DPMI32__                11
    __TURBOC__                10
    __WATCOMC__               10
    defined          10+     11+
    delay            14
    dos              12
    h       12
    msec             14
```

```
/* +++Date last modified: 05-Jul-1997 */


/*************************************************************
 *
 *  File : DEQUE.c
 *
 *  Author: Peter Yard [1993.01.02] -- 02 Jan 1993
 *
 *  Disclaimer: This code is released to the public domain.
 *
 *  Description:
 *      Generic double ended queue (Deque pronounced DEK) for handling
 *      any data types, with sorting.
 *
 *      By use of various functions in this module the caller
 *      can create stacks, queues, lists, doubly linked lists,
 *      sorted lists, indexed lists.  All lists are dynamic.
 *
 *      It is the responsibility of the caller to malloc and free
 *      memory for insertion into the queue. A pointer to the object
 *      is used so that not only can any data be used but various kinds
 *      of data can be pushed on the same queue if one so wished e.g.
 *      various length string literals mixed with pointers to structures
 *      or integers etc.
 *
 *  Enhancements:
 *      A future improvement would be the option of multiple "cursors"
 *      so that multiple locations could occur in the one queue to allow
 *      placemarkers and additional flexibility.  Perhaps even use queue
 *      itself to have a list of cursors.
 *
 * Usage:
 *
 *          /x init queue x/
 *          queue  q;
 *          Q_Init(&q);
 *
 *      To create a stack :
 *
 *          Q_PushHead(&q, &mydata1); /x push x/
 *          Q_PushHead(&q, &mydata2);
 *          .....
 *          data_ptr = Q_PopHead(&q); /x pop x/
 *          .....
 *          data_ptr = Q_First(&q);   /x top of stack x/
 *
 *      To create a FIFO:
 *
 *          Q_PushHead(&q, &mydata1);
 *          .....
 *          data_ptr = Q_PopTail(&q);
 *
 *      To create a double list:
 *
 *          data_ptr = Q_First(&q);
 *          ....
 *          data_ptr = Q_Next(&q);
 *          data_ptr = Q_Last(&q);
 *          if (Q_Empty(&q)) ....
 *          .....
 *          data_ptr = Q_Previous(&q);
 *
 *      To create a sorted list:
 *
 *          Q_PushHead(&q, &mydata1); /x push x/
 *          Q_PushHead(&q, &mydata2);
 *          .....
 *          if (!Q_Sort(&q, MyFunction))
 *              .. error ..
 *
 *          /x fill in key field of mydata1.
 *           * NB: Q_Find does linear search
 *           x/
 *
 *          if (Q_Find(&q, &mydata1, MyFunction))
 *          {
 *              /x found it, queue cursor now at correct record x/
 *              /x can retrieve with x/
 *              data_ptr = Q_Get(&q);
 *
 *              /x alter data , write back with x/
 *              Q_Put(&q, data_ptr);
 *          }
 *
 *          /x Search with binary search x/
 *          if (Q_Seek(&q, &mydata, MyFunction))
 *              /x etc x/
 *
 *
 *************************************************************/


#include <stdlib.h>

#include "deque.h"


static void QuickSort(void *list[], int low, int high,
                      int (*Comp)(const void *, const void *));
```

```
101 | static int  Q_BSearch(queue *q, void *key,
102 |                       int (*Comp)(const void *, const void *));
103 |
104 | /* The index: a pointer to pointers */
105 |
106 | static  void        **index;
107 | static  datanode    **posn_index;
108 |
109 |
110 | /***
111 |  *
112 |  ** function    : Q_Init
113 |  *
114 |  ** purpose     : Initialise queue object and pointers.
115 |  *
116 |  ** parameters  : 'queue' pointer.
117 |  *
118 |  ** returns     : True_ if init successful else False_
119 |  *
120 |  ** comments    :
121 |  ***/
122 |
123 | int Q_Init(queue  *q)
124 | {
125 |       q->head = q->tail = NULL;
126 |       q->cursor = q->head;
127 |       q->size = 0;
128 |       q->sorted = False_;
129 |
130 |       return True_;
131 | }
132 |
133 | /***
134 |  *
135 |  ** function    : Q_Start
136 |  *
137 |  ** purpose     : tests if cursor is at head of queue
138 |  *
139 |  ** parameters  : 'queue' pointer.
140 |  *
141 |  ** returns     : boolean - True_ is at head else False_
142 |  *
143 |  ** comments    :
144 |  *
145 |  ***/
146 |
147 | int Q_Start(queue *q)
148 | {
149 |       return (q->cursor == q->head);
150 | }
151 |
152 |
153 | /***
154 |  *
155 |  ** function    : Q_End
156 |  *
157 |  ** purpose     : boolean test if cursor at tail of queue
158 |  *
159 |  ** parameters  : 'queue' pointer to test.
160 |  *
161 |  ** returns     : True_ or False_
162 |  *
163 |  ** comments    :
164 |  *
165 |  ***/
166 |
167 | int Q_End(queue *q)
168 | {
169 |       return (q->cursor == q->tail);
170 | }
171 |
172 |
173 | /***
174 |  *
175 |  ** function    : Q_Empty
176 |  *
177 |  ** purpose     : test if queue has nothing in it.
178 |  *
179 |  ** parameters  : 'queue' pointer
180 |  *
181 |  ** returns     : True_ if empty queue, else False_
182 |  *
183 |  ** comments    :
184 |  *
185 |  ***/
186 |
187 | int Q_Empty(queue *q)
188 | {
189 |       return (q->size == 0);
190 | }
191 |
192 | /***
193 |  *
194 |  ** function    : Q_Size
195 |  *
196 |  ** purpose     : return the number of elements in the queue
197 |  *
198 |  ** parameters  : queue pointer
199 |  *
200 |  ** returns     : number of elements
```

```
201 |  *
202 |  ** comments    :
203 |  *
204 |  ***/
205 |
206 | int Q_Size(queue *q)
207 | {
208 |       return q->size;
209 | }
210 |
211 |
212 | /***
213 |  *
214 |  ** function    : Q_First
215 |  *
216 |  ** purpose     : position queue cursor to first element (head) of queue.
217 |  *
218 |  ** parameters  : 'queue' pointer
219 |  *
220 |  ** returns     : pointer to data at head. If queue is empty returns NULL
221 |  *
222 |  ** comments    :
223 |  *
224 |  ***/
225 |
226 | void *Q_First(queue *q)
227 | {
228 |       if (Q_Empty(q))
229 |             return NULL;
230 |
231 |       q->cursor = q->head;
232 |
233 |       return  q->cursor->data;
234 | }
235 |
236 |
237 | /***
238 |  *
239 |  ** function    : Q_Last
240 |  *
241 |  ** purpose     : locate cursor at tail of queue.
242 |  *
243 |  ** parameters  : 'queue' pointer
244 |  *
245 |  ** returns     : pointer to data at tail , if queue empty returns NULL
246 |  *
247 |  ** comments    :
248 |  *
249 |  ***/
250 |
251 | void *Q_Last(queue *q)
252 | {
253 |       if (Q_Empty(q))
254 |             return NULL;
255 |
256 |       q->cursor = q->tail;
257 |
258 |       return  q->cursor->data;
259 | }
260 |
261 |
262 | /***
263 |  *
264 |  ** function    : Q_PushHead
265 |  *
266 |  ** purpose     : put a data pointer at the head of the queue
267 |  *
268 |  ** parameters  : 'queue' pointer, void pointer to the data.
269 |  *
270 |  ** returns     : True_ if success else False_ if unable to push data.
271 |  *
272 |  ** comments    :
273 |  *
274 |  ***/
275 |
276 | int Q_PushHead(queue *q, void *d)
277 | {
278 |       node    *n;
279 |       datanode *p;
280 |
281 |       q->head->prev = malloc(sizeof(datanode));
282 |       if (q->head->prev == NULL)
283 |             return False_;
284 |
285 |       n = q->head;
286 |
287 |       p = q->head->prev;
288 |       q->head = (node*)p;
289 |       q->head->prev = NULL;
290 |
291 |       if (q->size == 0)
292 |       {
293 |             q->head->next = NULL;
294 |             q->tail = q->head;
295 |       }
296 |       else  q->head->next = (datanode*)n;
297 |
298 |       q->head->data = d;
299 |       q->size++;
300 |
```

```
301 |       q->cursor = q->head;
302 |
303 |       q->sorted = False_;
304 |
305 |       return True_;
306 | }
307 |
308 |
309 |
310 | /***
311 |  *
312 |  ** function    : Q_PushTail
313 |  *
314 |  ** purpose     : put a data element pointer at the tail of the queue
315 |  *
316 |  ** parameters  : queue pointer, pointer to the data
317 |  *
318 |  ** returns     : True_ if data pushed, False_ if data not inserted.
319 |  *
320 |  ** comments    :
321 |  *
322 |  ***/
323 |
324 | int Q_PushTail(queue *q, void *d)
325 | {
326 |       node        *p;
327 |       datanode    *n;
328 |
329 |       q->tail->next = malloc(sizeof(datanode));
330 |       if (q->tail->next == NULL)
331 |             return False_;
332 |
333 |       p = q->tail;
334 |       n = q->tail->next;
335 |       q->tail = (node *)n;
336 |
337 |       if (q->size == 0)
338 |       {
339 |             q->tail->prev = NULL;
340 |             q->head = q->tail;
341 |       }
342 |       else  q->tail->prev = (datanode *)p;
343 |
344 |       q->tail->next = NULL;
345 |
346 |       q->tail->data =  d;
347 |       q->cursor = q->tail;
348 |       q->size++;
349 |
350 |       q->sorted = False_;
351 |
352 |       return True_;
353 | }
354 |
355 |
356 |
357 | /***
358 |  *
359 |  ** function    : Q_PopHead
360 |  *
361 |  ** purpose     : remove and return the top element at the head of the
362 |  *                 queue.
363 |  *
364 |  ** parameters  : queue pointer
365 |  *
366 |  ** returns     : pointer to data element or NULL if queue is empty.
367 |  *
368 |  ** comments    :
369 |  *
370 |  ***/
371 |
372 | void *Q_PopHead(queue *q)
373 | {
374 |       datanode    *n;
375 |       void        *d;
376 |
377 |       if (Q_Empty(q))
378 |             return NULL;
379 |
380 |       d = q->head->data;
381 |       n = q->head->next;
382 |       free(q->head);
383 |
384 |       q->size--;
385 |
386 |       if (q->size == 0)
387 |             q->head = q->tail = q->cursor = NULL;
388 |       else
389 |       {
390 |             q->head = (node *)n;
391 |             q->head->prev = NULL;
392 |             q->cursor = q->head;
393 |       }
394 |
395 |       q->sorted = False_;
396 |
397 |       return d;
398 | }
399 |
400 |
```

```
401 | /***
402 |  *
403 |  ** function    : Q_PopTail
404 |  *
405 |  ** purpose     : remove element from tail of queue and return data.
406 |  *
407 |  ** parameters  : queue pointer
408 |  *
409 |  ** returns     : pointer to data element that was at tail. NULL if queue
410 |  *                 empty.
411 |  *
412 |  ** comments    :
413 |  *
414 |  ***/
415 |
416 | void *Q_PopTail(queue *q)
417 | {
418 |       datanode    *p;
419 |       void        *d;
420 |
421 |       if (Q_Empty(q))
422 |             return NULL;
423 |
424 |       d = q->tail->data;
425 |       p = q->tail->prev;
426 |       free(q->tail);
427 |       q->size--;
428 |
429 |       if (q->size == 0)
430 |             q->head = q->tail = q->cursor = NULL;
431 |       else
432 |       {
433 |             q->tail = (node *)p;
434 |             q->tail->next = NULL;
435 |             q->cursor = q->tail;
436 |       }
437 |
438 |       q->sorted = False_;
439 |
440 |       return d;
441 | }
442 |
443 |
444 |
445 | /***
446 |  *
447 |  ** function    : Q_Next
448 |  *
449 |  ** purpose     : Move to the next element in the queue without popping
450 |  *
451 |  ** parameters  : queue pointer.
452 |  *
453 |  ** returns     : pointer to data element of new element or NULL if end
454 |  *                 of the queue.
455 |  *
456 |  ** comments    : This uses the cursor for the current position. Q_Next
457 |  *                 only moves in the direction from the head of the queue
458 |  *                 to the tail.
459 |  ***/
460 |
461 | void *Q_Next(queue *q)
462 | {
463 |       if (q->cursor->next == NULL)
464 |             return NULL;
465 |
466 |       q->cursor = (node *)q->cursor->next;
467 |
468 |       return  q->cursor->data ;
469 | }
470 |
471 |
472 |
473 | /***
474 |  *
475 |  ** function    : Q_Previous
476 |  *
477 |  ** purpose     : Opposite of Q_Next. Move to next element closer to the
478 |  *                 head of the queue.
479 |  *
480 |  ** parameters  : pointer to queue
481 |  *
482 |  ** returns     : pointer to data of new element else NULL if queue empty
483 |  *
484 |  ** comments    : Makes cursor move towards the head of the queue.
485 |  *
486 |  ***/
487 |
488 | void *Q_Previous(queue *q)
489 | {
490 |       if (q->cursor->prev == NULL)
491 |             return NULL;
492 |
493 |       q->cursor = (node *)q->cursor->prev;
494 |
495 |       return q->cursor->data;
496 | }
497 |
498 |
499 |
500 | /***
```

```
 *
 ** function    : Q_DelCur
 *
 ** purpose     : Delete the current queue element as pointed to by
 *                the cursor.
 *
 ** parameters  : queue pointer
 *
 ** returns     : pointer to data element.
 *
 ** comments    : WARNING! It is the responsibility of the caller to
 *                free any memory. Queue cannot distinguish between
 *                pointers to literals and malloced memory.
 *
 ***/

void *Q_DelCur(queue *q)
{
      void        *d;
      datanode    *n, *p;

      if (q->cursor == NULL)
            return NULL;

      if (q->cursor == q->head)
            return Q_PopHead(q);

      if (q->cursor == q->tail)
            return Q_PopTail(q);

      n = q->cursor->next;
      p = q->cursor->prev;
      d = q->cursor->data;

      free(q->cursor);
      if (p != NULL)
            q->cursor = p;
      else  q->cursor = n;
      q->size--;

      q->sorted = False_;

      return d;
}



/***
 *
 ** function    : Q_Get
 *
 ** purpose     : get the pointer to the data at the cursor location
 *
 ** parameters  : queue pointer
 *
 ** returns     : data element pointer
 *
 ** comments    :
 *
 ***/

void *Q_Get(queue *q)
{
      if (q->cursor == NULL)
            return NULL;
      return q->cursor->data;
}



/***
 *
 ** function    : Q_Put
 *
 ** purpose     : replace pointer to data with new pointer to data.
 *
 ** parameters  : queue pointer, data pointer
 *
 ** returns     : boolean- True_ if successful, False_ if cursor at NULL
 *
 ** comments    :
 *
 ***/

int Q_Put(queue *q, void *data)
{
      if (q->cursor == NULL)
            return False_;

      q->cursor->data = data;
      return True_;
}


/***
 *
 ** function    : Q_Find
 *
 ** purpose     : Linear search of queue for match with key in *data
 *
```

```
 ** parameters  : queue pointer q, data pointer with data containing key
 *                comparison function here called Comp.
 *
 ** returns     : True_ if found , False_ if not in queue.
 *
 ** comments    : Useful for small queues that are constantly changing
 *                and would otherwise need constant sorting with the
 *                Q_Seek function.
 *                For description of Comp see Q_Sort.
 *                Queue cursor left on position found item else at end.
 *
 ***/

int Q_Find(queue *q, void *data,
               int (*Comp)(const void *, const void *))
{
      void *d;
      d = Q_First(q);
      do
      {
            if (Comp(d, data) == 0)
                  return True_;
            d = Q_Next(q);
      } while (!Q_End(q));

      if (Comp(d, data) == 0)
            return True_;

      return False_;
}

/*========  Sorted Queue and Index functions   ========= */


static void QuickSort(void *list[], int low, int high,
                        int (*Comp)(const void *, const void *))
{
      int     flag = 1, i, j;
      void    *key, *temp;

      if (low < high)
      {
            i = low;
            j = high + 1;

            key = list[ low ];

            while (flag)
            {
                  i++;
                  while (Comp(list[i], key) < 0)
                        i++;

                  j--;
                  while (Comp(list[j], key) > 0)
                        j--;

                  if (i < j)
                  {
                        temp = list[i];
                        list[i] = list[j];
                        list[j] = temp;
                  }
                  else  flag = 0;
            }

            temp = list[low];
            list[low] = list[j];
            list[j] = temp;

            QuickSort(list, low, j-1, Comp);
            QuickSort(list, j+1, high, Comp);
      }
}


/***
 *
 ** function    : Q_Sort
 *
 ** purpose     : sort the queue and allow index style access.
 *
 ** parameters  : queue pointer, comparison function compatible with
 *                with 'qsort'.
 *
 ** returns     : True_ if sort succeeded. False_ if error occurred.
 *
 ** comments    : Comp function supplied by caller must return
 *                   -1 if data1  < data2
 *                    0 if data1 == data2
 *                   +1 if data1  > data2
 *
 *                     for Comp(data1, data2)
 *
 *                If queue is already sorted it frees the memory of the
 *                old index and starts again.
 *
 ***/

int Q_Sort(queue *q, int (*Comp)(const void *, const void *))
```

```
{
      int         i;
      void        *d;
      datanode    *dn;

      /* if already sorted free memory for tag array */

      if (q->sorted)
      {
            free(index);
            free(posn_index);
            q->sorted = False_;
      }

      /* Now allocate memory of array, array of pointers */

      index = malloc(q->size * sizeof(q->cursor->data));
      if (index == NULL)
            return False_;

      posn_index = malloc(q->size * sizeof(q->cursor));
      if (posn_index == NULL)
      {
            free(index);
            return False_;
      }

      /* Walk queue putting pointers into array */

      d = Q_First(q);
      for (i=0; i < q->size; i++)
      {
            index[i] = d;
            posn_index[i] = q->cursor;
            d = Q_Next(q);
      }

      /* Now sort the index */

      QuickSort(index, 0, q->size - 1, Comp);

      /* Rearrange the actual queue into correct order */

      dn = q->head;
      i = 0;
      while (dn != NULL)
      {
            dn->data = index[i++];
            dn = dn->next;
      }

      /* Re-position to original element */

      if (d != NULL)
            Q_Find(q, d, Comp);
      else  Q_First(q);

      q->sorted = True_;

      return True_;
}


/***
 *
 ** function    : Q_BSearch
 *
 ** purpose     : binary search of queue index for node containing key
 *
 ** parameters  : queue pointer 'q', data pointer of key 'key',
 *                  Comp comparison function.
 *
 ** returns     : integer index into array of node pointers,
 *                or -1 if not found.
 *
 ** comments    : see Q_Sort for description of 'Comp' function.
 *
 ***/

static int Q_BSearch( queue *q, void *key,
                      int (*Comp)(const void *, const void*))
{
      int low, mid, hi, val;

      low = 0;
      hi = q->size - 1;

      while (low <= hi)
      {
            mid = (low + hi) / 2;
            val = Comp(key, index[ mid ]);

            if (val < 0)
                  hi = mid - 1;

            else if (val > 0)
                  low = mid + 1;

            else /* Success */
                  return mid;
```

```
801 |       }
802 |
803 |       /* Not Found */
804 |
805 |       return -1;
806 | }
807 |
808 |
809 | /***
810 |  *
811 |  ** function    : Q_Seek
812 |  *
813 |  ** purpose     : use index to locate data according to key in 'data'
814 |  *
815 |  ** parameters  : queue pointer 'q', data pointer 'data', Comp comparison
816 |  *                 function.
817 |  *
818 |  ** returns     : pointer to data or NULL if could not find it or could
819 |  *                 not sort queue.
820 |  *
821 |  ** comments    : see Q_Sort for description of 'Comp' function.
822 |  *
823 |  ***/
824 |
825 | void *Q_Seek(queue *q, void *data, int (*Comp)(const void *, const void *))
826 | {
827 |       int idx;
828 |
829 |       if (!q->sorted)
830 |       {
831 |             if (!Q_Sort(q, Comp))
832 |                   return NULL;
833 |       }
834 |
835 |       idx = Q_BSearch(q, data, Comp);
836 |
837 |       if (idx < 0)
838 |             return NULL;
839 |
840 |       q->cursor = posn_index[idx];
841 |
842 |       return index[idx];
843 | }
844 |
845 |
846 |
847 | /***
848 |  *
849 |  ** function    : Q_Insert
850 |  *
851 |  ** purpose     : Insert an element into an indexed queue
852 |  *
853 |  ** parameters  : queue pointer 'q', data pointer 'data', Comp comparison
854 |  *                 function.
855 |  *
856 |  ** returns     : pointer to data or NULL if could not find it or could
857 |  *                 not sort queue.
858 |  *
859 |  ** comments    : see Q_Sort for description of 'Comp' function.
860 |  *                 WARNING! This code can be very slow since each new
861 |  *                 element means a new Q_Sort.  Should only be used for
862 |  *                 the insertion of the odd element ,not the piecemeal
863 |  *                 building of an entire queue.
864 |  ***/
865 |
866 | int Q_Insert(queue *q, void *data, int (*Comp)(const void *, const void *))
867 | {
868 |       Q_PushHead(q, data);
869 |
870 |       if (!Q_Sort(q, Comp))
871 |             return False_;
872 |
873 |       return True_;
874 | }
```

TEXT STATISTICS

18585 characters
874 lines

LEXICAL STATISTICS

33 comments [std-C]
0 comments [C++]
2 preprocessor instructions
0 constants [character]
1 constants [string]
28 constants [numeric]

SYMBOL TABLE

```
stdlib          94
tail           125     169     256     294     329     330     333     334     335     339     340
       342     344     346     347     387     424     425     426     430     433     434     435
       528
temp           639     660     662     667     669
val            783     791     793     796
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  File : DEQUE.H
 *
 *  Peter Yard  02 Jan 1993.
 */

#ifndef DEQUEUE__H
#define DEQUEUE__H

#include "sniptype.h"                    /* For True_, False_    */


typedef struct nodeptr datanode;

typedef struct nodeptr {
      void        *data ;
      datanode    *prev, *next ;
} node ;

typedef struct {
      node        *head, *tail, *cursor;
      int         size, sorted, item_deleted;
} queue;

typedef  struct {
      void        *dataptr;
      node        *loc ;
} index_elt ;


int    Q_Init(queue  *q);
int    Q_Empty(queue *q);
int    Q_Size(queue *q);
int    Q_Start(queue *q);
int    Q_End(queue *q);
int    Q_PushHead(queue *q, void *d);
int    Q_PushTail(queue *q, void *d);
void  *Q_First(queue *q);
void  *Q_Last(queue *q);
void  *Q_PopHead(queue *q);
void  *Q_PopTail(queue *q);
void  *Q_Next(queue *q);
void  *Q_Previous(queue *q);
void  *Q_DelCur(queue *q);
void  *Q_Get(queue *q);
int    Q_Put(queue *q, void *data);
int    Q_Sort(queue *q, int (*Comp)(const void *, const void *));
int    Q_Find(queue *q, void *data,
               int (*Comp)(const void *, const void *));
void  *Q_Seek(queue *q, void *data,
               int (*Comp)(const void *, const void *));
int    Q_Insert(queue *q, void *data,
                int (*Comp)(const void *, const void *));

#endif /* DEQUEUE__H */
```

TEXT STATISTICS

```
  1419 characters
    57 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Comp            49      51      53      55
    DEQUEUE__H               9      10
    Q_DelCur        46
    Q_Empty         34
    Q_End           37
    Q_Find          50
    Q_First         40
    Q_Get           47
    Q_Init          33
    Q_Insert        54
    Q_Last          41
    Q_Next          44
    Q_PopHead       42
    Q_PopTail       43
    Q_Previous              45
    Q_PushHead              38
    Q_PushTail              39
    Q_Put           48
    Q_Seek          52
    Q_Size          35
    Q_Sort          49
    Q_Start         36
    cursor          23
    d       38      39
    data            18      48      50      52      54
    datanode        15      19
    dataptr         28
    head            23
    index_elt       30
    item_deleted            24
    loc             29
    next            19
    node            20      23      29
    nodeptr         15      17
    prev            19
    q       33      34      35      36      37      38      39      40      41      42      43      44
            45      46      47      48      49      50      52      54
    queue           25      33      34      35      36      37      38      39      40      41      42
            43      44      45      46      47      48      49      50      52      54
    size            24
    sorted          24
    tail            23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DIRENT.H - Posix compliant header
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#ifndef DIRENT_H
#define DIRENT_H

#include <stdio.h>                      /* For FILENAME_MAX     */
#include <dos.h>
#include "sniptype.h"
#include "dirport.h"

#undef DIR
#define DIR DIR_

/*
**  Posix directory structure
*/

typedef struct {
      int               dd_fd;
      unsigned          dd_loc,
                        dd_size;
      DOSFileData       dd_buf;
      char              dd_dirname[FILENAME_MAX];
} DIR;

DIR            *opendir(char *);
int             closedir(DIR *),
                rewinddir(DIR *);
struct dirent  *readdir(DIR *),
               *seekdir(DIR *, int, int);
#define         telldir(dd) dd->loc

/*
**  Other useful functions from DIRMASK.C and XSTRCMP.C
*/

int       dirmask(DOSFileData *, char *, char *, unsigned, unsigned);
Boolean_T xstrcmp (const char *, const char *);
Boolean_T xstricmp (const char *, const char *);

extern int DFerr;

extern DIR _DIRS[];

#endif /* DIRENT_H */
```

TEXT STATISTICS

```
  1381 characters
    56 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       49      50
    DFerr           52
    DIR             22      23      35      37      38      39      40      41      54
    DIRENT_H        14      15
    DIR_            23
    DOSFileData             33      48
    FILENAME_MAX            34
    _DIRS           54
    closedir        38
    dd              42+
    dd_buf          33
    dd_dirname              34
    dd_fd           30
    dd_loc          31
    dd_size         32
    dirent          40
    dirmask         48
    dos             18
    h       17      18
    loc             42
    opendir         37
    readdir         40
    rewinddir       39
    seekdir         41
    stdio           17
    telldir         42
    xstrcmp         49
    xstricmp        50
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DIRMASK.C - Complex pattern matching
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdio.h>
#include <string.h>
#include "sniptype.h"
#include "dirent.h"

/*************************************************************/
/*                                                           */
/*  dirmask()                                                */
/*                                                           */
/*  Tests a directory entry for matching patterns. Tests both     */
/*  file name and attributes. Tests for both inclusion specs      */
/*  and exclusion specs.                                     */
/*                                                           */
/*  Parameters: 1 - Pointer to the directory entry's FIND        */
/*                  structure                                */
/*              2 - Filename for inclusion matching, i.e. if     */
/*                  this spec matches the filename, we matched.  */
/*                  Use NULL to match anything.              */
/*              3 - Filename for exclusion matching, i.e. if     */
/*                  this spec matches the filename, we failed.   */
/*                  Use NULL to exclude nothing.             */
/*              4 - Attribute for inclusion mask. Use FA_ANY     */
/*                  to match anything).                      */
/*              5 - Attribute for exclusion mask. Use zero to    */
/*                  exclude nothing).                        */
/*                                                           */
/*  Returns: Success_ if name and attribute matched,             */
/*           else Error_.                                    */
/*                                                           */
/*  Side effects: Converts patterns to upper case                */
/*                                                           */
/*************************************************************/

int dirmask(DOSFileData    *dstruct,
            char           *fname_inc,
            char           *fname_exc,
            unsigned        attr_inc,
            unsigned        attr_exc)
{
      if (!dstruct)
            return Error_;
      strupr(fname_inc);
      strupr(fname_exc);
      if (fname_inc)
      {
            if (True_ != xstrcmp(ff_name(dstruct), fname_inc))
                  return Error_;
      }
      if (fname_exc)
      {
            if (True_ == xstrcmp(ff_name(dstruct), fname_exc))
                  return Error_;
      }
      if (!(ff_attr(dstruct) & attr_inc))
            return Error_;
      if (ff_attr(dstruct) & attr_exc)
            return Error_;
      return Success_;
}
```

```
TEXT STATISTICS

  3124 characters
    72 lines

LEXICAL STATISTICS

    29 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    DOSFileData             47
    Error_          54      60      65      68      70
    Success_        71
    True_           59      64
    attr_exc        51      69
    attr_inc        50      67
    dirmask         47
    dstruct         47      53      59      64      67      69
    ff_attr         67      69
    ff_name         59      64
    fname_exc       49      56      62      64
    fname_inc       48      55      57      59
    h       14      15
    stdio           14
    string          15
    strupr          55      56
    xstrcmp         59      64
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  WIN 32 & OS/2 support functions for DIRPORT.H
**
**  Public domain by Jerry Coffin
**
**  15-May-96 David Nugent    Moved OS/2 functions from
**                            dirport.h to here
**                            Fixed for 32-bit OS/2
**                            Allowed compilation under
**                            DOS for easier makefiles
*/

#if defined(__MSDOS__) || defined(MSDOS)
/* #error "DirPort.c is NOT intended for DOS programs ..." */
#else

#include "dirport.h"

#if defined(OS2)

int FIND_FIRST (char * spec, unsigned attr, DOSFileData *ff)
{
  _SYSINT cnt = 1;
  ff->dh = (HDIR) -1;
  return (int) DosFindFirst ((PSZ)spec, &ff->dh, (_SYSINT)attr, &ff->f,
                             (_SYSINT)sizeof(struct _FILEFINDBUF), &cnt, FINDLVL);
}

int FIND_NEXT (DOSFileData *ff)
{
  _SYSINT cnt = 1;
  return (int) DosFindNext (ff->dh, &ff->f, sizeof(struct _FILEFINDBUF), &cnt);
}

int FIND_END (DOSFileData *ff)
{
  return (int) DosFindClose (ff->dh);
}

#else   /* WIN32 & WINNT */

WORD ff_date(DOSFileData *f)
{
      WORD DOS_date;
      WORD DOS_time;

      FileTimeToDosDateTime(&(f->file.ftLastWriteTime), &DOS_date, &DOS_time);

      return DOS_date;
}

WORD ff_time(DOSFileData *f)
{
      WORD DOS_date;
      WORD DOS_time;

      FileTimeToDosDateTime(&(f->file.ftLastWriteTime), &DOS_date, &DOS_time);

      return DOS_time;
}

WORD ff_yr(DOSFileData *f)
{
      SYSTEMTIME t;

      FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);

      return t.wYear;
}

WORD ff_mo(DOSFileData *f)
{
      SYSTEMTIME t;

      FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);

      return t.wMonth;
}


WORD ff_day(DOSFileData *f)
{
      SYSTEMTIME t;

      FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);

      return t.wDay;
}

WORD ff_hr(DOSFileData*f)
{
      SYSTEMTIME t;

      FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);

      return t.wHour;
}

```

```
101 | WORD ff_min(DOSFileData *f)
102 | {
103 |       SYSTEMTIME t;
104 |
105 |       FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);
106 |
107 |       return t.wMinute;
108 | }
109 |
110 | WORD ff_tsec(DOSFileData *f)
111 | {
112 |       SYSTEMTIME t;
113 |
114 |       FileTimeToSystemTime(&(f->file.ftLastWriteTime), &t);
115 |
116 |       return t.wMilliseconds / 100;
117 | }
118 |
119 | int FIND_FIRST(char *spec, unsigned attrib, DOSFileData *ff)
120 | {
121 |       ff_attr(ff) = attrib | _A_ARCH;
122 |
123 |       ff->handle = FindFirstFile(spec, &(ff->file));
124 |
125 |       if ( INVALID_HANDLE_VALUE == ff->handle ) {
126 |             return 1;
127 |       }
128 |
129 |       while ( 0 != (ff->file.dwFileAttributes & ~ff_attr(ff)))
130 |       {
131 |             if ( FALSE == FindNextFile(ff->handle, &(ff->file))) {
132 |                   return 1;
133 |             }
134 |       }
135 |       return 0;
136 | }
137 |
138 | int FIND_NEXT(DOSFileData *ff)
139 | {
140 |       do {
141 |             if ( FALSE == FindNextFile(ff->handle, &(ff->file))) {
142 |                   return 1;
143 |             }
144 |
145 |       } while ( 0 != (ff->file.dwFileAttributes & ~ff_attr(ff)));
146 |       return 0;
147 | }
148 |
149 | int FIND_END(DOSFileData *ff)
150 | {
151 |     return (int)FindClose(ff->handle);
152 | }
153 |
154 | #endif /* OS/2   */
155 |
156 | #endif /* MS-DOS */
```

```
TEXT STATISTICS

  3207 characters
   156 lines

LEXICAL STATISTICS

     6 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     1 constants [string]
    11 constants [numeric]

SYMBOL TABLE

    DOSFileData             23      31      37      44      54      64      73      83      92     101
           110     119     138     149
    DOS_date                46      49      51      56      59
    DOS_time                47      49      57      59      61
    DosFindClose            39
    DosFindFirst            27
    DosFindNext             34
    FALSE                  131     141
    FINDLVL                 28
    FIND_END                37     149
    FIND_FIRST              23     119
    FIND_NEXT               31     138
    FileTimeToDosDateTime           49      59
    FileTimeToSystemTime            68      77      87      96     105     114
    FindClose              151
    FindFirstFile          123
    FindNextFile           131     141
    HDIR                    26
    INVALID_HANDLE_VALUE           125
    MSDOS                   15
    OS2                     21
    PSZ                     27
    SYSTEMTIME              66      75      85      94     103     112
    WORD                    44      46      47      54      56      57      64      73      83      92     101
           110
    _A_ARCH                121
    _FILEFINDBUF            28      34
    _SYSINT                 25      27      28      33
    __MSDOS__               15
    attr                    23      27
    attrib                 119     121
    cnt                     25      28      33      34
    defined                 15+     21
    dh                      26      27      34      39
    dwFileAttributes       129     145
    f       27      34      44      49      54      59      64      68      73      77      83      87
            92      96     101     105     110     114
    ff                      23      26      27+     31      34+     37      39     119     121     123+    125
           129+    131+    138     141+    145+    149     151
    ff_attr                121     129     145
    ff_date                 44
    ff_day                  83
    ff_hr                   92
    ff_min                 101
    ff_mo                   73
    ff_time                 54
    ff_tsec                110
    ff_yr                   64
    file                    49      59      68      77      87      96     105     114     123     129     131
           141     145
    ftLastWriteTime         49      59      68      77      87      96     105     114
    handle                 123     125     131     141     151
    spec                    23      27     119     123
    t       66      68      70      75      77      79      85      87      89      94      96      98
           103     105     107     112     114     116
    wDay                    89
    wHour                   98
    wMilliseconds          116
    wMinute                107
    wMonth                  79
    wYear                   70
```

```
/* +++Date last modified: 05-Jul-1997 */

/*======================================================================

    _MSC_VER        Microsoft C 6.0 and later
    _QC             Microsoft Quick C 2.51 and later
    __TURBOC__      Borland Turbo C, Turbo C++ and BC++
    __BORLANDC__    Borland C++
    __ZTC__         Zortech C and C++
    __SC__          Symantec C++
    __WATCOMC__     WATCOM C
    __POWERC        Mix Power C
    __GNUC__        Gnu C
    __EMX__         Emx Gnu C

    Revised:

    Original   Scott Ladd   Now obsolete
    14-Sep-93  Fred Cole    Moved MK_FP() macro to end of file to avoid
                            redefinition error when dos.h gets included
                            at the in/outport definitions for __TURBOC__
    15-Sep-93  Thad Smith   Add conditional code for TC 2.01
                            Fix findfirst/findnext support for ZTC 3.0
    15-Oct-93  Bob Stout    Revise find first/next support
    02-Dec-93  David Nugent Additions for findfirst/findnext support for
                            MSC6 (& 7) for OS/2
                            Added FIND_END macro for use under OS/2 to
                            be nice about closing the directory handle
                            DOSFileData members should be accessed via
                            the new ff_*() macros for portability
                            Note: use -DOS2 when compiling under OS/2
    03-Apr-94  Bob Stout    Add Power C support, FAR
    19-Aug-95  Bob Stout    Add NEAR, PASCAL, CDECL, and portable attributes
    06-Sep-95  Phi Nguyen   Add DOSFileTime, DOSFileDate, & supporting macros
               Bob Stout    (ff_yr/mo/day(), ff_hr/min/tsec())
    08-Sep-95  Bob Nelson   Add __GNUCC__ and PAK macro
    21-Sep-95  Bob Stout    Renamed to PC-PORT.H, revised directory stuff
    25-Sep-95  Bob Stout    Split out EXTKWORD.H
    26-Sep-95  Jerry Coffin Added Win32 support.
    21-Oct-95  Bob Stout    Resolve struct dirent incompatibilities,
                            move port I/O macros to PCHWIO.H,
                            move MK_FP() cover to SNIPTYPE.H,
                            rename to DIRPORT.H(!!!)
    15-May-96  David Nugent Added 32-bit OS/2 support (Watcom/emx)
    04-Jul-96  Bob Stout    Fixed attribute redefinition problems w/ TC 3.x
    28-Jul-96  Bob Stout    Fixed error in definition of ff_sec() in DOS
    20-Aug-96  Bob Stout    Eliminate Win32 conflicts
======================================================================*/


/* prevent multiple inclusions of this header file */

#if !defined(DIRPORT__H)
#define DIRPORT__H

#if defined(__OS2__)
 #define OS2
#endif

#if defined(__EMX__)
 #undef OS2
 #define OS2
#endif

#if defined(OS2)
 #undef MSDOS
#endif

#include "extkword.h"         /* Extended keywords header   */

#undef dirent
#define dirent DIRENT_

/*
**  Correct file attributes
*/

#if defined(__POWERC) || (defined(__TURBOC__) && !defined(__BORLANDC__))
 #ifndef FA_NORMAL
  #define FA_NORMAL 0x00
 #endif

 #undef  _A_NORMAL
 #undef  _A_RDONLY
 #undef  _A_HIDDEN
 #undef  _A_SYSTEM
 #undef  _A_VOLID
 #undef  _A_SUBDIR
 #undef  _A_ARCH
 #undef  _A_ANY

 #define _A_NORMAL      FA_NORMAL
 #define _A_RDONLY      FA_RDONLY
 #define _A_HIDDEN      FA_HIDDEN
 #define _A_SYSTEM      FA_SYSTEM
 #define _A_VOLID       FA_LABEL
 #define _A_SUBDIR      FA_DIREC
 #define _A_ARCH        FA_ARCH
 #define _A_ANY   FA_NORMAL | FA_RDONLY | FA_HIDDEN | FA_SYSTEM | FA_LABEL | \
                  FA_DIREC | FA_ARCH
```

```
#else
 #ifdef OS2
  #define _A_NORMAL     FILE_NORMAL
  #define _A_RDONLY     FILE_READONLY
  #define _A_HIDDEN     FILE_HIDDEN
  #define _A_SYSTEM     FILE_SYSTEM
  #define _A_SUBDIR     FILE_DIRECTORY
  #define _A_VOLID      0
  #define _A_ARCH       FILE_ARCHIVED
  #define _A_ANY  FILE_NORMAL | FILE_READONLY | FILE_HIDDEN | FILE_SYSTEM | \
                  FILE_DIRECTORY | FILE_ARCHIVED
 #elif defined(_WIN32) || defined(WIN32)
  #define WIN32_LEAN_AND_MEAN
  #define NOGDI
  #define NOSERVICE
  #undef INC_OLE1
  #undef INC_OLE2
  #include <windows.h>
  #include <dos.h>    /* #define's most _A_* file attribute macros    */

  #define _A_VOLID   0
  #define _A_ANY     _A_NORMAL | _A_RDONLY | _A_HIDDEN | _A_SYSTEM | \
     _A_SUBDIR | _A_ARCH

  #define ff_name(x)     (x)->file.cFileName
  #define ff_size(x)     (x)->file.nFileSizeLow
  #define ff_attr(x)     (x)->file.dwFileAttributes

  typedef struct dirent {
     WIN32_FIND_DATA file;
     HANDLE handle;
  } DOSFileData;

  WORD ff_date(DOSFileData *f);
  WORD ff_yr(DOSFileData *f);
  WORD ff_mo(DOSFileData *f);
  WORD ff_day(DOSFileData *f);
  WORD ff_time(DOSFileData *f);
  WORD ff_hr(DOSFileData *f);
  WORD ff_min(DOSFileData *f);
  WORD ff_tsec(DOSFileData *f);

  int FIND_FIRST(char *spec, unsigned attr, DOSFileData *ff);
  int FIND_NEXT(DOSFileData *ff);
  int FIND_END(DOSFileData *ff);
 #elif defined(__MSDOS__) || defined(MSDOS)
  #define _A_ANY  _A_NORMAL | _A_RDONLY | _A_HIDDEN | _A_SYSTEM | _A_VOLID | \
                  _A_SUBDIR | _A_ARCH
 #endif
#endif

/*---------------------------------------------------------------------
    Directory search macros and data structures

    DOSFileData         MS-DOS file data structure
    DOSFileDate         MS-DOS file date structure
    DOSFileTime         MS-DOS file time structure
    FIND_FIRST          MS-DOS function 0x4E -- find first matching spec
    FIND_NEXT           MS-DOS function 0x4F -- find subsequent files
---------------------------------------------------------------------*/

/* make sure the structure is packed on byte boundary */

#if defined(_MSC_VER) || defined(_QC) || defined(__WATCOMC__)
    #pragma pack(1)
#elif defined(__ZTC__)
    #pragma ZTC align 1
#elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
    #pragma option -a-
#endif

#ifdef __GNUC__
 #define PAK      __attribute__((packed))
#else
 #define PAK
#endif

/*
**  Structures to access dates and times
*/

typedef struct {
            unsigned tsecs:   5;
            unsigned mins:    6;
            unsigned hours:   5;
      } DOSFileTime;

typedef struct {
            unsigned day:     5;
            unsigned month:   4;
            unsigned year:    7;
      } DOSFileDate;

/*
**  Use this structure in place of compiler-defined file structure.
**  Always use the ff_xxxx() macros to access structure members.
*/

#if defined(OS2)
 #define INCL_DOS
```

```
201 |  #include "os2.h"
202 |  #undef TRUE
203 |  #undef FALSE
204 |  #undef  _SYSINT
205 |  #if defined(__FLAT__) || defined(__EMX__)
206 |  #undef  _FILEFINDBUF
207 |  #define _FILEFINDBUF _FILEFINDBUF3
208 |  #define _SYSINT ULONG
209 |  #define FINDLVL FIL_STANDARD
210 |  #else
211 |  #define _SYSINT USHORT
212 |  #define FINDLVL 0L
213 |  #endif
214 |
215 |  typedef struct dirent {
216 |       HDIR dh;
217 |       struct _FILEFINDBUF f;
218 |       } DOSFileData;
219 |
220 |  #define d_name         achName     /* For struct dirent portability    */
221 |  #define d_size         cbFile
222 |
223 |  #define ff_name(x)     (x)->f.d_name
224 |  #define ff_size(x)     (x)->f.d_size
225 |  #define ff_attr(x)     (x)->f.attrFile
226 |  #define ff_date(x)     *((DOSFileDate *)(&(x)->f.fdateLastWrite))
227 |  #define ff_yr(x)       *((DOSFileDate *)(&(x)->f.fdateLastWrite)).year
228 |  #define ff_mo(x)       *((DOSFileDate *)(&(x)->f.fdateLastWrite)).month
229 |  #define ff_day(x)      *((DOSFileDate *)(&(x)->f.fdateLastWrite)).day
230 |  #define ff_time(x)     *((DOSFileTime *)(&(x)->f.ftimeLastWrite))
231 |  #define ff_hr(x)       *((DOSFileTime *)(&(x)->f.ftimeLastWrite)).hours
232 |  #define ff_min(x)      *((DOSFileTime *)(&(x)->f.ftimeLastWrite)).mins
233 |  #define ff_tsec(x)     *((DOSFileTime *)(&(x)->f.ftimeLastWrite)).tsecs
234 | #elif !(defined(WIN32) || defined(_WIN32)) /* Not OS/2  or Win32 */
235 |
236 |  typedef struct dirent {
237 |       char             reserved[21];
238 |       char             attrib;
239 |       DOSFileTime      time;
240 |       DOSFileDate      date;
241 |       long             d_size;
242 |       char             d_name[13];
243 |       } DOSFileData;
244 |
245 |  #define ff_name(x)     (x)->d_name
246 |  #define ff_size(x)     (x)->d_size
247 |  #define ff_attr(x)     (x)->attrib
248 |  #define ff_date(x)     (x)->date
249 |  #define ff_yr(x)       (x)->date.year
250 |  #define ff_mo(x)       (x)->date.month
251 |  #define ff_day(x)      (x)->date.day
252 |  #define ff_time(x)     (x)->time
253 |  #define ff_hr(x)       (x)->time.hours
254 |  #define ff_min(x)      (x)->time.mins
255 |  #define ff_sec(x)      (x)->time.tsecs
256 | #endif
257 |
258 | /* set structure alignment to default */
259 |
260 | #if defined (_MSC_VER) || defined(_QC) || defined(__WATCOMC__)
261 |  #pragma pack()
262 | #elif defined (__ZTC__)
263 |  #pragma ZTC align
264 | #elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
265 |  #pragma option -a.
266 | #endif
267 |
268 | /* include proper header files and create macros */
269 |
270 | #if defined (_MSC_VER) || defined(_QC) || defined(__WATCOMC__) || defined(__EMX__)
271 |  #if defined( OS2 )
272 |   int FIND_FIRST (char * spec, unsigned attr, DOSFileData *ff);
273 |   int FIND_NEXT (DOSFileData *ff);
274 |   int FIND_END (DOSFileData *ff);
275 |
276 |  #elif !(defined(_WIN32) || defined(WIN32))
277 |   #include "dos.h"
278 |   #define FIND_FIRST(spec,attr,buf) _dos_findfirst(spec,attr, \
279 |         (struct find_t *)buf)
280 |   #define FIND_NEXT(buf) _dos_findnext((struct find_t *)buf)
281 |   #define FIND_END(buf)
282 |  #endif
283 | #elif defined (__TURBOC__) || defined (__BORLANDC__) || defined (__POWERC)
284 |  #include "dir.h"
285 |  #include "dos.h"
286 |  #define FIND_FIRST(spec,attr,buf) findfirst(spec,(struct ffblk *)buf,attr)
287 |  #define FIND_NEXT(buf) findnext((struct ffblk *)buf)
288 |  #define FIND_END(buf)
289 | #elif defined(__SC__) || defined(__WATCOMC__) || defined(__ZTC__)
290 |  #include "dos.h"
291 |  #define FIND_FIRST(spec,attr,buf) _dos_findfirst(spec,attr, \
292 |        (struct find_t *)buf)
293 |  #define FIND_NEXT(buf) _dos_findnext((struct find_t *)buf)
294 |  #define FIND_END(buf)
295 | #endif
296 |
297 | #endif /* DIRPORT__H */
```

TEXT STATISTICS

9922 characters
297 lines

LEXICAL STATISTICS

15 comments [std-C]
0 comments [C++]
143 preprocessor instructions
0 constants [character]
6 constants [string]
16 constants [numeric]

SYMBOL TABLE

```
    DIRENT_          72
    DIRPORT__H               53      54
    DOSFileData             132     134     135     136     137     138     139     140     141     143
          144     145     218     243     272     273     274
    DOSFileDate              192     226     227     228     229     240
    DOSFileTime              186     230     231     232     233     239
    FALSE           203
    FA_ARCH          98     100
    FA_DIREC         97     100
    FA_HIDDEN        94      99
    FA_LABEL         96      99
    FA_NORMAL        79      80      92      99
    FA_RDONLY        93      99
    FA_SYSTEM        95      99
    FILE_ARCHIVED            109     111
    FILE_DIRECTORY           107     111
    FILE_HIDDEN              105     110
    FILE_NORMAL              103     110
    FILE_READONLY            104     110
    FILE_SYSTEM              106     110
    FIL_STANDARD             209
    FINDLVL         209     212
    FIND_END        145     274     281     288     294
    FIND_FIRST               143     272     278     286     291
    FIND_NEXT       144     273     280     287     293
    HANDLE          131
    HDIR            216
    INCL_DOS        200
    INC_OLE1        116
    INC_OLE2        117
    MSDOS            66     146
    NOGDI           114
    NOSERVICE       115
    OS2              57      61      62      65     102     199     271
    PAK             173     175
    TRUE            202
    ULONG           208
    USHORT          211
    WIN32           112     234     276
    WIN32_FIND_DATA          130
    WIN32_LEAN_AND_MEAN      113
    WORD            134     135     136     137     138     139     140     141
    ZTC             167     263
    _A_ANY           90      99     110     122     147
    _A_ARCH          89      98     109     123     148
    _A_HIDDEN        85      94     105     122     147
    _A_NORMAL        83      92     103     122     147
    _A_RDONLY        84      93     104     122     147
    _A_SUBDIR        88      97     107     123     148
    _A_SYSTEM        86      95     106     122     147
    _A_VOLID         87      96     108     121     147
    _FILEFINDBUF             206     207     217
    _FILEFINDBUF3            207
    _MSC_VER        164     260     270
    _QC             164     260     270
    _SYSINT         204     208     211
    _WIN32          112     234     276
    __BORLANDC__              78     283
    __EMX__          60     205     270
    __FLAT__        205
    __GNUC__        172
    __MSDOS__       146
    __OS2__          56
    __POWERC         78     283
    __SC__          289
    __TURBOC__                78     168+    264+    283
    __WATCOMC__              164     260     270     289
    __ZTC__         166     262     289
    __attribute__            173
    _dos_findfirst           278     291
    _dos_findnext            280     293
    a       169     265
    achName         220
    align           167     263
    attr            143     272     278+    286+    291+
    attrFile        225
    attrib          238     247
    buf             278     279     280+    281     286+    287+    288     291     292     293+    294
    cFileName       125
    cbFile          221
    d_name          220     223     242     245
    d_size          221     224     241     246
    date            240     248     249     250     251
    day             189     229     251
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DISK_SN.C - Retrieve a disk serial number
**
**  public domain demo by Paul Schlyter and Bob Stout
**  OS/2 compatibility added by Mark Kimes
*/

#include <stdio.h>

#ifdef OS2
 #define INCL_DOS

 #include <os2.h>
 #include <stdlib.h>
#else
 #include <dos.h>
#endif

#include "extkword.h"
#include "disk_sn.h"
#include "mk_fp.h"

/*
**  Arguments: 1 - Drive (0 = default, 1 = A:, 2 = B:, etc.)
**
**  Returns: Static string containing the disk serial number, or
**           an empty string if no serial number is available,
**
**  Notes: Drive errors result in an empty string returned, this function
**         does not invoke the critical error handler.
*/

char *get_disk_id(int drive)
{
      static char serial_no[10];
#ifndef OS2
      union REGS r;
      struct SREGS s;
      unsigned sno1, sno2;

      r.x.ax = 0x6900;
      r.h.bl = drive;
      segread(&s);
      intdosx(&r, &r, &s);
      if (r.x.cflag)
            *serial_no = '\0';
      else
      {
            sno2 = *((unsigned FAR *)MK_FP(s.ds, r.x.dx+2));
            sno1 = *((unsigned FAR *)MK_FP(s.ds, r.x.dx+4));
            sprintf(serial_no, "%04X-%04X\n", sno1, sno2);
      }
#else
      struct {
            ULONG serial;
            char  volumelength;
            char  volumelabel[CCHMAXPATH];
      } volser;

      DosError(FERR_DISABLEHARDERR);
      if(0 == DosQueryFSInfo(drive, FSIL_VOLSER, &volser, sizeof(volser)))
            sprintf(serial_no, "%04X-%04X", HIUSHORT(volser.serial),
                  LOUSHORT(volser.serial));
      else  *serial_no = 0;
#endif
      return serial_no;
}

#ifdef TEST

#include <ctype.h>

main(int argc, char *argv[])
{
      char *sn;

      if (2 > argc)
      {
            puts("Usage: DISK_SN drive_letter [...drive_letter]");
            return -1;
      }
      while (--argc)
      {
            int drive;

            drive = toupper(**++argv);
            sn = get_disk_id(drive - '@');
            printf("The serial number of %c: is %s\n", drive, sn);
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2182 characters
    95 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
    17 preprocessor instructions
     2 constants [character]
     7 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    CCHMAXPATH              59
    DosError        62
    DosQueryFSInfo          63
    FAR             51      52
    FERR_DISABLEHARDERR             62
    FSIL_VOLSER             63
    HIUSHORT        64
    INCL_DOS        13
    LOUSHORT        65
    MK_FP           51      52
    OS2             12      38
    REGS            39
    SREGS           40
    TEST            71
    ULONG           57
    argc            75      79      84
    argv            75      88
    ax              43
    bl              44
    cflag           47
    ctype           73
    dos             18
    drive           35      44      63      86      88      89      90
    ds              51      52
    dx              51      52
    get_disk_id             35      89
    h       10      15      16      18      44      73
    intdosx         46
    main            75
    os2             15
    printf          90
    puts            81
    r       39      43      44      46+     47      51      52
    s       40      45      46      51      52
    segread         45
    serial          57      64      65
    serial_no       37      48      53      64      66      68
    sn              77      89      90
    sno1            41      52      53
    sno2            41      51      53
    sprintf         53      64
    stdio           10
    stdlib          16
    toupper         88
    volser          60      63+     64      65
    volumelabel             59
    volumelength            58
    x       43      47      51      52
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  get_disk_id() - Retrieve a disk serial number
   5 | */
   6 |
   7 | #ifndef DISK_SN__H
   8 | #define DISK_SN__H
   9 |
  10 | char *get_disk_id(int drive);
  11 |
  12 | #endif /*  DISK_SN__H */


TEXT STATISTICS

   206 characters
    12 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    DISK_SN__H               7       8
    drive           10
    get_disk_id             10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DO.C - a simple facility for specifying multiple commands
  5 | */
  6 |
  7 | #include <stdio.h>
  8 | #include <stdlib.h>
  9 |
 10 | main(int argc, char *argv[])
 11 | {
 12 |       if (2 > argc)
 13 |       {
 14 |             puts("Usage: DO \"DOS command 1\" \"DOS command 2\" ...");
 15 |             return -1;
 16 |       }
 17 |       while (--argc)
 18 |             system(*++argv);
 19 |       return 0;
 20 | }
```

TEXT STATISTICS

```
   400 characters
    20 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    argc            10      12      17
    argv            10      18
    h       7        8
    main            10
    puts            14
    stdio            7
    stdlib           8
    system          18
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  DOANSI.H - Portable ANSI screen code interpreter
   5 | **
   6 | **  From DRSK_105.LZH (ansi.c), hereby declared free for use for whatever
   7 | **  purposes by author: Mark Kimes
   8 | */
   9 |
  10 | #ifndef DOANSI__H
  11 | #define DOANSI__H
  12 |
  13 | extern char curattr;
  14 | extern int  curx, cury;
  15 | extern int  maxx, maxy;
  16 | extern int  realmaxy,realmaxx;
  17 | extern char useansi;
  18 | extern int  tabspaces;
  19 |
  20 | /* set maxx,maxy as desired */
  21 | void set_screensize (int reservedlines);
  22 |
  23 | /* put character c at x,y using attr as attribute */
  24 | void put_char (char c,char attr,int x,int y);
  25 |
  26 | /* position hardware cursor at x,y */
  27 | void pos_hardcursor (int x,int y);
  28 |
  29 | /* turn hardware cursor off */
  30 | void hardcursor_off (void);
  31 |
  32 | /* turn hardware cursor on at x,y */
  33 | void hardcursor_on (int x,int y);
  34 |
  35 | /* scroll window tx,ty - bx,by up one line; fill with blank+attr */
  36 | void scroll_up (int tx,int ty,int bx,int by,char attr);
  37 |
  38 | /* clear the window from tx,ty - bx,by; fill with blank+attr */
  39 | void clearwindow (int tx,int ty,int bx,int by,char attr);
  40 |
  41 | /* clear line y from col x to eol (ex); fill with blank+attr */
  42 | void cleartoeol (int x,int y,int ex,char attr);
  43 |
  44 | /* the ansi string interpreter */
  45 | int ansi_out (char *buf);
  46 |
  47 | #endif /* DOANSI__H */
```

TEXT STATISTICS

```
  1267 characters
    47 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DOANSI__H       10      11
    ansi_out        45
    attr            24      36      39      42
    buf             45
    bx              36      39
    by              36      39
    c       24
    cleartoeol              42
    clearwindow             39
    curattr         13
    curx            14
    cury            14
    ex              42
    hardcursor_off          30
    hardcursor_on           33
    maxx            15
    maxy            15
    pos_hardcursor          27
    put_char        24
    realmaxx        16
    realmaxy        16
    reservedlines           21
    scroll_up       36
    set_screensize          21
    tabspaces       18
    tx              36      39
    ty              36      39
    useansi         17
    x       24      27      33      42
    y       24      27      33      42
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DOANSI_1.C - Portable ANSI screen code interpreter
**
**  From DRSK_105.LZH (ansi.c), hereby declared free for use for whatever
**  purposes by author: Mark Kimes
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "doansi.h"

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif

#ifdef __TURBOC__   /* shut up 'Parameter not used' warnings        */
 #pragma warn -par  /* in the BC3.1 IDE the setting of Options,     */
#endif              /* Compiler,Messages,Display must NOT be 'All'. */

/*
 * to initialize:
 *   call set_screensize(<# lines to reserve>);
 * to print through ansi interpreter:
 *   call ansi_out(<string>);
 */

char               curattr = 7;
int                curx = 0,cury = 0;
int                maxx = 80, maxy = 25;  /* size of ansi output window */
int                realmaxy,realmaxx;     /* real screen size */
char               useansi = 1;           /* while true, interp ansi seqs */
int                tabspaces = 8;
static int         savx,savy,issaved = 0;
static char        ansi_terminators[] = "HFABCDnsuJKmp";

#define MAXARGLEN       128

#define NOTHING         0
#define WASESCAPE       1
#define WASBRKT         2

/* "generic" support functions closely related to ansi_out */

void set_pos (char *argbuf,int arglen,char cmd)
{
      int   y,x;
      char *p;

      if (!*argbuf || !arglen)
      {
            curx = cury = 0;
      }
      y = atoi(argbuf) - 1;
      p = strchr(argbuf,';');
      if (y >= 0 && p)
      {
            x = atoi(p + 1) - 1;
            if(x >= 0)
            {
                  curx = x;
                  cury = y;
            }
      }
}

void go_up (char *argbuf,int arglen,char cmd)
{
      int x;

      x = atoi(argbuf);
      if (!x)
            x = 1;
      for ( ; x ; x--)
      {
            if (!cury)
                  break;
            cury--;
      }
}

void go_down (char *argbuf,int arglen,char cmd)
{
      int x;

      x = atoi(argbuf);
      if (!x)
            x = 1;
      for ( ; x ; x--)
      {
            if (cury == maxy - 1)
                  break;
            cury++;
      }
}

void go_left (char *argbuf,int arglen,char cmd)
{
```

```
101 |       int x;
102 |
103 |       x = atoi(argbuf);
104 |       if (!x)
105 |             x = 1;
106 |       for ( ; x ; x--)
107 |       {
108 |             if(!curx)
109 |                   break;
110 |             curx--;
111 |       }
112 | }
113 |
114 | void go_right (char *argbuf,int arglen,char cmd)
115 | {
116 |       int x;
117 |
118 |       x = atoi(argbuf);
119 |       if (!x)
120 |             x = 1;
121 |       for ( ; x ; x--)
122 |       {
123 |             if (curx == maxx - 1)
124 |                   break;
125 |             curx++;
126 |       }
127 | }
128 |
129 | void report (char *argbuf,int arglen,char cmd)
130 | {
131 |       /* you figure out how to implement it ... */
132 | }
133 |
134 | void save_pos (char *argbuf,int arglen,char cmd)
135 | {
136 |       savx = curx;
137 |       savy = cury;
138 |       issaved = 1;
139 | }
140 |
141 | void restore_pos (char *argbuf,int arglen,char cmd)
142 | {
143 |       if(issaved)
144 |       {
145 |             curx = savx;
146 |             cury = savy;
147 |             issaved = 0;
148 |       }
149 | }
150 |
151 | void clear_screen (char *argbuf,int arglen,char cmd)
152 | {
153 |       /* needs error checking */
154 |
155 |       clearwindow(0,0,maxx - 1,maxy - 1,curattr);
156 |       curx = cury = 0;
157 | }
158 |
159 | void kill_line (char *argbuf,int arglen,char cmd)
160 | {
161 |       cleartoeol(curx,cury,maxx - 1,curattr);
162 | }
163 |
164 |
165 | void set_colors (char *argbuf,int arglen,char cmd)
166 | {
167 |       char *p,*pp;
168 |
169 |       if (*argbuf && arglen)
170 |       {
171 |             pp = argbuf;
172 |             do
173 |             {
174 |                   p = strchr(pp,';');
175 |                   if (p && *p)
176 |                   {
177 |                         *p = 0;
178 |                         p++;
179 |                   }
180 |                   switch (atoi(pp))
181 |                   {
182 |                   case 0: /* all attributes off */
183 |                         curattr = 7;
184 |                         break;
185 |
186 |                   case 1: /* bright on */
187 |                         curattr |= 8;
188 |                         break;
189 |
190 |                   case 2: /* faint on */
191 |                         curattr &= (~8);
192 |                         break;
193 |
194 |                   case 3: /* italic on */
195 |                         break;
196 |
197 |                   case 5: /* blink on */
198 |                         curattr |= 128;
199 |                         break;
200 |
```

```
201 |                   case 6: /* rapid blink on */
202 |                         break;
203 |
204 |                   case 7: /* reverse video on */
205 |                         curattr = 112;
206 |                         break;
207 |
208 |                   case 8: /* concealed on */
209 |                         curattr = 0;
210 |                         break;
211 |
212 |                   case 30: /* black fg */
213 |                         curattr &= (~7);
214 |                         break;
215 |
216 |                   case 31: /* red fg */
217 |                         curattr &= (~7);
218 |                         curattr |= 4;
219 |                         break;
220 |
221 |                   case 32: /* green fg */
222 |                         curattr &= (~7);
223 |                         curattr |= 2;
224 |                         break;
225 |
226 |                   case 33: /* yellow fg */
227 |                         curattr &= (~7);
228 |                         curattr |= 6;
229 |                         break;
230 |
231 |                   case 34: /* blue fg */
232 |                         curattr &= (~7);
233 |                         curattr |= 1;
234 |                         break;
235 |
236 |                   case 35: /* magenta fg */
237 |                         curattr &= (~7);
238 |                         curattr |= 5;
239 |                         break;
240 |
241 |                   case 36: /* cyan fg */
242 |                         curattr &= (~7);
243 |                         curattr |= 3;
244 |                         break;
245 |
246 |                   case 37: /* white fg */
247 |                         curattr |= 7;
248 |                         break;
249 |
250 |                   case 40: /* black bg */
251 |                         curattr &= (~112);
252 |                         break;
253 |
254 |                   case 41: /* red bg */
255 |                         curattr &= (~112);
256 |                         curattr |= (4 << 4);
257 |                         break;
258 |
259 |                   case 42: /* green bg */
260 |                         curattr &= (~112);
261 |                         curattr |= (2 << 4);
262 |                         break;
263 |
264 |                   case 43: /* yellow bg */
265 |                         curattr &= (~112);
266 |                         curattr |= (6 << 4);
267 |                         break;
268 |
269 |                   case 44: /* blue bg */
270 |                         curattr &= (~112);
271 |                         curattr |= (1 << 4);
272 |                         break;
273 |
274 |                   case 45: /* magenta bg */
275 |                         curattr &= (~112);
276 |                         curattr |= (5 << 4);
277 |                         break;
278 |
279 |                   case 46: /* cyan bg */
280 |                         curattr &= (~112);
281 |                         curattr |= (3 << 4);
282 |                         break;
283 |
284 |                   case 47: /* white bg */
285 |                         curattr |= 112;
286 |                         break;
287 |
288 |                   case 48: /* subscript bg */
289 |                         break;
290 |
291 |                   case 49: /* superscript bg */
292 |                         break;
293 |
294 |                   default: /* unsupported */
295 |                         break;
296 |                   }
297 |                   pp = p;
298 |             } while (p);
299 |       }
300 | }
```

```
301 |
302 | int ansi_out (char *buf)
303 | {
304 |       int  arglen = 0, ansistate = NOTHING, x;
305 |       char *b = buf, argbuf[MAXARGLEN] = "";
306 |
307 |       /* cursor is off while string is being displayed so we don't have
308 |          to keep updating it.  works to our detriment only if using
309 |          BIOS writes under MS-DOS
310 |       */
311 |
312 |       hardcursor_off();
313 |
314 |       if (!useansi)                     /* is ANSI interp on? */
315 |       {
316 |             ansistate = NOTHING;
317 |             arglen = 0;
318 |             *argbuf = 0;
319 |       }
320 |
321 |       while (*b)
322 |       {
323 |             switch (ansistate)
324 |             {
325 |             case NOTHING:
326 |                   switch (*b)
327 |                   {
328 |                   case '\x1b':
329 |                         if (useansi)
330 |                         {
331 |                               ansistate = WASESCAPE;
332 |                               break;
333 |                         }
334 |
335 |                   case '\r':
336 |                         curx = 0;
337 |                         break;
338 |
339 |                   case '\n':
340 |                         cury++;
341 |                         if (cury > maxy - 1)
342 |                         {
343 |                               scroll_up(0,0,maxx - 1,maxy - 1,curattr);
344 |                               cury--;
345 |                         }
346 |                         break;
347 |
348 |                   case '\t':     /* so _you_ figure out what to do... */
349 |                         for (x = 0; x < tabspaces; x++)
350 |                         {
351 |                               put_char(' ',curattr,curx,cury);
352 |                               curx++;
353 |                               if (curx > maxx - 1)
354 |                               {
355 |                                     curx = 0;
356 |                                     cury++;
357 |                                     if (cury > maxy - 1)
358 |                                     {
359 |                                           scroll_up(0, 0, maxx - 1, maxy - 1,
360 |                                                 curattr);
361 |                                           cury--;
362 |                                     }
363 |                               }
364 |                         }
365 |                         break;
366 |
367 |                   case '\b':
368 |                         if (curx)
369 |                         {
370 |                               curx--;
371 |                         }
372 |                         break;
373 |
374 |                   case '\07':     /* usually a console bell */
375 |                         putchar('\07');
376 |                         break;
377 |
378 |                   default:
379 |                         put_char(*b,curattr,curx,cury);
380 |                         curx++;
381 |                         if (curx > maxx - 1)
382 |                         {
383 |                               curx = 0;
384 |                               cury++;
385 |                               if (cury > maxy - 1)
386 |                               {
387 |                                     scroll_up(0,0,maxx - 1,maxy - 1,curattr);
388 |                                     cury--;
389 |                               }
390 |                         }
391 |                         break;
392 |                   }
393 |                   break;
394 |
395 |             case WASESCAPE:
396 |                   if (*b == '[')
397 |                   {
398 |                         ansistate = WASBRKT;
399 |                         arglen = 0;
400 |                         *argbuf = 0;
```

```
401 |                         break;
402 |                   }
403 |                   ansistate = NOTHING;
404 |                   break;
405 |
406 |             case WASBRKT:
407 |                   if (strchr(ansi_terminators, (int)*b))
408 |                   {
409 |                         switch ((int)*b)
410 |                         {
411 |                         case 'H':   /* set cursor position */
412 |                         case 'F':
413 |                               set_pos(argbuf,arglen,*b);
414 |                               break;
415 |
416 |                         case 'A':   /* up */
417 |                               go_up(argbuf,arglen,*b);
418 |                               break;
419 |
420 |                         case 'B':   /* down */
421 |                               go_down(argbuf,arglen,*b);
422 |                               break;
423 |
424 |                         case 'C':   /* right */
425 |                               go_right(argbuf,arglen,*b);
426 |                               break;
427 |
428 |                         case 'D':   /* left */
429 |                               go_left(argbuf,arglen,*b);
430 |                               break;
431 |
432 |                         case 'n':   /* report pos */
433 |                               report(argbuf,arglen,*b);
434 |                               break;
435 |
436 |                         case 's':   /* save pos */
437 |                               save_pos(argbuf,arglen,*b);
438 |                               break;
439 |
440 |                         case 'u':   /* restore pos */
441 |                               restore_pos(argbuf,arglen,*b);
442 |                               break;
443 |
444 |                         case 'J':   /* clear screen */
445 |                               clear_screen(argbuf,arglen,*b);
446 |                               break;
447 |
448 |                         case 'K':   /* delete to eol */
449 |                               kill_line(argbuf,arglen,*b);
450 |                               break;
451 |
452 |                         case 'm':   /* set video attribs */
453 |                               set_colors(argbuf,arglen,*b);
454 |                               break;
455 |
456 |                         case 'p':   /* keyboard redef -- disallowed */
457 |                               break;
458 |
459 |                         default:    /* unsupported */
460 |                               break;
461 |                         }
462 |                         ansistate = NOTHING;
463 |                         arglen = 0;
464 |                         *argbuf = 0;
465 |                   }
466 |                   else
467 |                   {
468 |                         if (arglen < MAXARGLEN)
469 |                         {
470 |                               argbuf[arglen] = *b;
471 |                               argbuf[arglen + 1] = 0;
472 |                               arglen++;
473 |                         }
474 |                   }
475 |                   break;
476 |
477 |             default:
478 |                   pos_hardcursor(curx,cury);
479 |                   fputs("\n **Error in ANSI state machine.\n",stderr);
480 |                   break;
481 |             }
482 |             b++;
483 |       }
484 |       pos_hardcursor(curx,cury);
485 |       hardcursor_on(curx,cury);
486 |
487 |       return ((int)b - (int)buf);
488 | }
```

TEXT STATISTICS

13841 characters
488 lines

LEXICAL STATISTICS

56 comments [std-C]
0 comments [C++]
14 preprocessor instructions
24 constants [character]
4 constants [string]
129 constants [numeric]

SYMBOL TABLE

```
MAXARGLEN       39     305     468
NOTHING         41     304     316     325     403     462
WASBRKT         43     398     406
WASESCAPE       42     331     395
__TURBOC__               19
__WATCOMC__              15
ansi_out       302
ansi_terminators         37     407
ansistate      304     316     323     331     398     403     462
argbuf          47      52      56      57      69      73      84      88      99     103     114
       118     129     134     141     151     159     165     169     171     305     318     400
       413     417     421     425     429     433     437     441     445     449     453     464
       470     471
arglen          47      52      69      84      99     114     129     134     141     151     159
       165     169     304     317     399     413     417     421     425     429     433     437
       441     445     449     453     463     468     470     471     472
atoi            56      60      73      88     103     118     180
b      305     321     326     379     396     407     409     413     417     421     425     429
       433     437     441     445     449     453     470     482     487
buf            302     305     487
clear_screen           151     445
cleartoeol             161
clearwindow            155
cmd             47      69      84      99     114     129     134     141     151     159     165
curattr         30     155     161     183     187     191     198     205     209     213     217
       218     222     223     227     228     232     233     237     238     242     243     247
       251     255     256     260     261     265     266     270     271     275     276     280
       281     285     343     351     360     379     387
curx            31      54      63     108     110     123     125     136     145     156     161
       336     351     352     353     355     368     370     379     380     381     383     478
       484     485
cury            31      54      64      78      80      93      95     137     146     156     161
       340     341     344     351     356     357     361     379     384     385     388     478
       484     485
fputs          479
go_down         84     421
go_left         99     429
go_right       114     425
go_up           69     417
h       10      11      12
hardcursor_off         312
hardcursor_on          485
issaved         36     138     143     147
kill_line      159     449
maxx            32     123     155     161     343     353     359     381     387
maxy            32      93     155     341     343     357     359     385     387
off             16
p       50      57      58      60     167     174     175+    177     178     297     298
par             20
pos_hardcursor         478     484
pp             167     171     174     180     297
put_char       351     379
putchar        375
realmaxx        33
realmaxy        33
report         129     433
restore_pos            141     441
save_pos       134     437
savx            36     136     145
savy            36     137     146
scroll_up      343     359     387
set_colors             165     453
set_pos         47     413
stderr         479
stdio           11
stdlib          10
strchr          57     174     407
string          12
tabspaces       35     349
unreferenced            16
useansi         34     314     329
warn            20
x       49      60      61      63      71      73      74      75      76+     86      88      89
        90      91+    101     103     104     105     106+    116     118     119     120     121+
       304     349+
y       49      56      58      64
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DOANSI_2.C - OS-Specific ANSI screen code interpreter functions
**
**  From DRSK_105.LZH (ansi.c), hereby declared free for use for whatever
**  purposes by author: Mark Kimes
*/

#include "doansi.h"
#include "mk_fp.h"

int   realmaxy, realmaxx;
int   maxx, maxy;

#ifdef OS2

#define INCL_DOS
#define INCL_VIO

#include <os2.h>

int vidhandle = 0;  /* can be changed for AVIO */

void set_screensize (int reservedlines)
{
      VIOMODEINFO vm;

      vm.cb = sizeof(VIOMODEINFO);
      VioGetMode(&vm, vidhandle);
      maxx = vm.col;
      maxy = vm.row - reservedlines;
      realmaxx = maxx;
      realmaxy = vm.row;
}

void pos_hardcursor (int x,int y)
{
      VioSetCurPos(y,x,vidhandle);
}

void hardcursor_off (void)
{
      VIOCURSORINFO vc;

      VioGetCurType(&vc,vidhandle);
      vc.attr = -1;
      VioSetCurType(&vc,vidhandle);
}

void hardcursor_on (int x,int y)
{
      VIOCURSORINFO vc;

      VioGetCurType(&vc,vidhandle);
      vc.attr = 0;
      VioSetCurType(&vc,vidhandle);
      VioSetCurPos(y,x,vidhandle);
}

void put_char (char c, char attr, int x, int y)
{
      VioWrtCharStrAtt(&c,1,y,x,&attr,vidhandle);
}

void scroll_up (int tx,int ty,int bx,int by,char attr)
{
      int attrib = ' ' | (attr << 8);

      VioScrollUp(ty,tx,by,bx,1,(char *)&attrib,vidhandle);
}

void clearwindow (int tx,int ty,int bx,int by,char attr)
{
      int attrib = ' ' | (attr << 8);

      VioScrollUp(ty,tx,by,bx,-1,(char *)&attrib,vidhandle);
}

void cleartoeol (int x,int y,int ex,char attr)
{
      int attrib = ' ' | (attr << 8);

      VioScrollUp(y,x,y,ex,-1,(char *)&attrib,vidhandle);
}

#else

/* MS-DOS -- (urp) */

#include <dos.h>

#if !defined(MK_FP)
 #define MK_FP(seg,off) ((void far *)(((long)(seg) << 16)|(unsigned)(off)))
#endif

static int far *vseg;
char           usebios = 0; /* if true, output through BIOS */

int vmode (void)
```

```
{
      union REGS r;

      r.h.ah = 15;
      r.x.bx = 0;
      int86(0x10,&r,&r);
      return r.h.al;
}

void set_screensize (int reservedlines)
{
      union REGS   r;
      unsigned int vbase;

      r.h.ah = 0x0f;
      r.x.bx = 0;
      int86 (0x10, &r, &r);
      maxx = (int) r.h.ah;
      if (maxx < 80)                                  /* gimme a break! */
      {
            r.x.ax = 0x0003;
            int86(0x10,&r,&r);
            maxx = 80;
      }
      realmaxx = maxx;
      r.x.ax = 0x1130;
      r.x.dx = maxy;
      int86 (0x10, &r, &r);
      realmaxy = maxy = (r.x.dx == 0) ? 25 : (r.x.dx + 1);
      maxy -= reservedlines;
      vbase = (vmode () == 7 ? 0xb000 : 0xb800);
      vseg = MK_FP(vbase,0);        /* address of video ram as pointer */
}

void pos_hardcursor (int x,int y)
{
      union REGS r;

      r.x.ax = 0x0200;
      r.x.bx = 0;
      r.x.dx = ((y << 8) & 0xff00) + x;
      int86(0x10,&r,&r);
}

void hardcursor_off (void)
{
      union REGS r;

      r.x.ax = 0x0200;
      r.x.bx = 0;
      r.x.dx = ((realmaxy << 8) & 0xff00);
      int86(0x10,&r,&r);
}

void hardcursor_on (int x,int y)
{
      union REGS r;

      r.x.ax = 0x0200;
      r.x.bx = 0;
      r.x.dx = ((y << 8) & 0xff00) + x;
      int86(0x10,&r,&r);
}

void put_char (char c, char attr, int x, int y)
{
      if(!usebios)
      {
            register int far *v;

             /* point v to right spot in vid RAM */

            v = vseg + ((y * realmaxx) + x);
            *v = (c | (attr << 8));                   /* display */
      }
      else
      {

            union REGS r;

            r.x.ax = 0x0200;
            r.x.bx = 0;
            r.x.dx = ((y << 8) & 0xff00) + x;
            int86(0x10,&r,&r);
            r.h.ah = 0x09;
            r.h.bh = 0;
            r.h.bl = attr;
            r.x.cx = 1;
            r.h.al = c;
            int86(0x10,&r,&r);
      }
}

void scroll_up (int tx,int ty,int bx,int by,char attr)
{
      union REGS r;

      r.h.ah = 6;
      r.h.al = 1;
      r.h.bh = attr;
```

```
201 |       r.h.cl = tx;
202 |       r.h.ch = ty;
203 |       r.h.dl = bx;
204 |       r.h.dh = by;
205 |       int86(0x10,&r,&r);
206 | }
207 |
208 | void clearwindow (int tx,int ty,int bx,int by,char attr)
209 | {
210 |       union REGS r;
211 |
212 |       r.h.ah = 6;
213 |       r.h.al = 0;
214 |       r.h.bh = attr;
215 |       r.h.cl = tx;
216 |       r.h.ch = ty;
217 |       r.h.dl = bx;
218 |       r.h.dh = by;
219 |       int86(0x10,&r,&r);
220 | }
221 |
222 | void cleartoeol (int x,int y,int ex,char attr)
223 | {
224 |       union REGS r;
225 |
226 |       r.h.ah = 6;
227 |       r.h.al = 0;
228 |       r.h.bh = attr;
229 |       r.h.cl = x;
230 |       r.h.ch = y;
231 |       r.h.dl = ex;
232 |       r.h.dh = y;
233 |       int86(0x10,&r,&r);
234 | }
235 |
236 | #endif
```

```
TEXT STATISTICS

  4764 characters
   236 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     3 constants [character]
     2 constants [string]
    65 constants [numeric]

SYMBOL TABLE

    INCL_DOS        18
    INCL_VIO        19
    MK_FP           93      94     132
    OS2             16
    REGS           102     112     137     147     157     179     196     210     224
    VIOCURSORINFO           44      53
    VIOMODEINFO             27      29
    VioGetCurType           46      55
    VioGetMode              30
    VioScrollUp             70      77      84
    VioSetCurPos            39      58
    VioSetCurType           48      57
    VioWrtCharStrAtt        63
    ah             104     115     118     185     198     212     226
    al             107     189     199     213     227
    attr            47      56      61      63      66      68      73      75      80      82     165
           174     187     194     200     208     214     222     228
    attrib          68      70      75      77      82      84
    ax             121     126     139     149     159     181
    bh             186     200     214     228
    bl             187
    bx              66      70      73      77     105     116     140     150     160     182     194
           203     208     217
    by              66      70      73      77     194     204     208     218
    c       61      63     165     174     189
    cb              29
    ch             202     216     230
    cl             201     215     229
    cleartoeol              80     222
    clearwindow             73     208
    col             31
    cx             188
    defined         93
    dh             204     218     232
    dl             203     217     231
    dos             91
    dx             127     129+    141     151     161     183
    ex              80      84     222     231
    far             94      97     169
    h       21      91     104     107     115     118     185     186     187     189     198     199
           200     201     202     203     204     212     213     214     215     216     217     218
           226     227     228     229     230     231     232
    hardcursor_off          42     145
    hardcursor_on           51     155
    int86          106     117     122     128     142     152     162     184     190     205     219
           233
    maxx            14      31      33     118     119     123     125
    maxy            14      32     127     129     130
    off             94+
    os2             21
    pos_hardcursor          37     135
    put_char        61     165
    r      102     104     105     106+    107     112     115     116     117+    118     121     122+
           126     127     128+    129+    137     139     140     141     142+    147     149     150
           151     152+    157     159     160     161     162+    179     181     182     183     184+
           185     186     187     188     189     190+    196     198     199     200     201     202
           203     204     205+    210     212     213     214     215     216     217     218     219+
           224     226     227     228     229     230     231     232     233+
    realmaxx        13      33     125     173
    realmaxy        13      34     129     151
    reservedlines           25      32     110     130
    row             32      34
    scroll_up       66     194
    seg             94+
    set_screensize          25     110
    tx              66      70      73      77     194     201     208     215
    ty              66      70      73      77     194     202     208     216
    usebios         98     167
    v      169     173     174
    vbase          113     131     132
    vc              44      46      47      48      53      55      56      57
    vidhandle       23      30      39      46      48      55      57      58      63      70      77
            84
    vm              27      29      30      31      32      34
    vmode          100     131
    vseg            97     132     173
    x       37      39      51      58      61      63      80      84     105     116     121     126
           127     129+    135     139     140     141+    149     150     151     155     159     160
           161+    165     173     181     182     183+    188     222     229
    y       37      39      51      58      61      63      80      84+     135     141     155     161
           165     173     183     222     230     232
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DOS5BOOT.H - DOS 5 boot record
  5 | */
  6 |
  7 | #ifndef DOS5BOOT__H
  8 | #define DOS5BOOT__H
  9 |
 10 | #include "extkword.h"
 11 |
 12 | #if defined(__TURBOC__)
 13 |  #pragma option -a-
 14 | #elif defined(__ZTC__)
 15 |  #pragma ZTC align 1
 16 | #else /* MSC/QC/WATCOM/METAWARE */
 17 |  #pragma pack(1)
 18 | #endif
 19 |
 20 | typedef struct {                                /* offset in buffer record */
 21 |         char      bsJump[3];                    /* 1 - 3                   */
 22 |         char      bsOemName[8];                 /* 4 - 11                  */
 23 |         short     bsBytesPerSec;                /* 12 - 13                 */
 24 |         char      bsSecPerClust;                /* 14                      */
 25 |         short     bsResSectors;                 /* 15 - 16                 */
 26 |         char      bsFATs;                       /* 17                      */
 27 |         short     bsRootDirEnts;                /* 18 - 19                 */
 28 |         short     bsSectors;                    /* 20 - 21                 */
 29 |         char      bsMedia;                      /* 22                      */
 30 |         short     bsFATsecs;                    /* 23 - 24                 */
 31 |         short     bsSecPerTrack;                /* 25 - 26                 */
 32 |         short     bsHeads;                      /* 27 - 28                 */
 33 |         long      bsHiddenSecs;                 /* 29 - 32                 */
 34 |         long      bsHugeSectors;                /* 33 - 36                 */
 35 |         char      bsDriveNumber;                /* 37                      */
 36 |         char      bsReserved1;                  /* 38                      */
 37 |         char      bsBootSignature;              /* 39                      */
 38 |         long      bsVolumeID;                   /* 40 - 43                 */
 39 |         char      bsVolumeLabel[11];            /* 44 - 54                 */
 40 |         char      bsFileSysType[8];             /* 54 - 61                 */
 41 |         char      bsReserved2[8];               /* 62 - 69                 */
 42 |         char      bsJunk[442];                  /* 70 - end of record
 43 |                                                    (byte 512 is last)      */
 44 |         } B_REC;                    /* Boot_record; total of 512 bytes     */
 45 |
 46 | #endif /* DOS5BOOT__H */
```

TEXT STATISTICS

```
  2310 characters
    46 lines
```

LEXICAL STATISTICS

```
    28 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     1 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    B_REC           44
    DOS5BOOT__H              7       8
    ZTC             15
    __TURBOC__              12
    __ZTC__         14
    a       13
    align           15
    bsBootSignature         37
    bsBytesPerSec           23
    bsDriveNumber           35
    bsFATs          26
    bsFATsecs       30
    bsFileSysType           40
    bsHeads         32
    bsHiddenSecs            33
    bsHugeSectors           34
    bsJump          21
    bsJunk          42
    bsMedia         29
    bsOemName       22
    bsResSectors            25
    bsReserved1             36
    bsReserved2             41
    bsRootDirEnts           27
    bsSecPerClust           24
    bsSecPerTrack           31
    bsSectors       28
    bsVolumeID              38
    bsVolumeLabel           39
    defined         12      14
    option          13
    pack            17
```

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************
 *      function : copy                          *
 *      purpose  : copy one file                 *
 *                                               *
 *      arguments: path to source 'fromDir',     *
 *                 path to target 'toDir',       *
 *                 filename to copy 'fname'      *
 *                                               *
 *      returns  : nothing                       *
 *                                               *
 *      By       :  Peter Yard (29 May 1991)     *
 *************************************************/

#include <stdlib.h>
#include <string.h>
#include <io.h>
#include "dosfiles.h"
#include "filnames.h"

#define STDOUT fileno(stdout)

void copy(char *fromDir, char *fname, char *toDir)
{
      FILE    *nul;       /* nul will redirect stdout to DOS 'nul' */
      char    from[FILENAME_MAX], to[FILENAME_MAX], comd[128];
      int     oldStdout;

      /* Create the strings to describe the paths                  */

      fnMerge(from, NULL, NULL, fromDir, NULL, fname, NULL);
      fnMerge(to, NULL, NULL, toDir, NULL, fname, NULL);

      /* Construct 'comd' string which is a dos command for a copy */

      strcpy(comd, "copy ");
      strcat(comd, from); strcat(comd, " ");
      strcat(comd, to);

      /* Redirect stdout to a nul file, kills output to the screen */

      nul = fopen("NUL", "w");
      oldStdout = dup(STDOUT);
      dup2(fileno(nul), STDOUT);
      fclose(nul);

      system(comd);           /* COPY file */

      /* Restore stdout and close nul file */

      dup2(oldStdout, STDOUT);
      close(oldStdout);

      /* Display file source and target,      */
      /* otherwise comment out the next line. */

      printf("\n%s copied to %s",from,to);
}
```

```
TEXT STATISTICS

  1923 characters
    59 lines

LEXICAL STATISTICS

    10 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     7 constants [string]
     1 constants [numeric]

SYMBOL TABLE

    FILE            26
    FILENAME_MAX            27+
    NULL            32+     33+
    STDOUT          22      44      45      52
    close           53
    comd            27      37      38+     39      48
    copy            24
    dup             44
    dup2            45      52
    fclose          46
    fileno          22      45
    fnMerge         32      33
    fname           24      32      33
    fopen           43
    from            27      32      38      58
    fromDir         24      32
    h       16      17      18
    io              18
    nul             26      43      45      46
    oldStdout       28      44      52      53
    printf          58
    stdlib          16
    stdout          22
    strcat          38+     39
    strcpy          37
    string          17
    system          48
    to              27      33      39      58
    toDir           24      33
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SNIPPETS header for functions to work with DOS files & directories
*/

#ifndef DOSFILES__H
#define DOSFILES__H

#include <stdio.h>
#include "dirport.h"
#include "sniptype.h"


Boolean_T   addpath(char *newdir);                    /* Addpath.C      */
void        copy(char *fromDir, char *fname,
                 char *toDir);                        /* Doscopy.C      */
int         getdrv(void);                             /* Drvalid.C      */
Boolean_T   chdrv(int drive);                         /* Drvalid.C      */
Boolean_T   drvalid(int drive);                       /* Drvalid.C      */
Boolean_T   drvrdy(int drive);                        /* Drvalid.C      */
int         favail(void);                             /* Favail.C       */
char      * getdcwd(unsigned int drive);              /* Getdcwd.C      */
int         iscons(FILE *fp);                         /* Iscons.C       */
int         isfopen(FILE *fp);                        /* Isfopen.C      */
int         drive_type(int dr);                       /* Isnetdr.C      */
Boolean_T   isRamDsk(unsigned char drive);            /* Isramdsk.C     */
int         isWprot(int drive);                       /* Iswprot.C      */
int         mkdirs(char *pathname);                   /* Mkdirs.C       */
int         PushDir(char *newdir);                    /* Pushdir.C      */
int         PopDir(void);                             /* Pushdir.C      */
int         isdir(char *dir);                         /* Pushdir.C      */
char FAR  * truename(char FAR *dst, char FAR *src);   /* Truename.C     */
Boolean_T   isCDROMdrive(int drive);                  /* Iscdrom.C      */
int         fdupdate(int fd);                         /* Fupdate.C      */
int         fupdate(FILE *fp);                        /* Fupdate.C      */

/*
**  File: ADDHNDLS.H
*/

#define TABLE_SIZE 255        /* NOTE: *Must* be <= FILES in CONFIG.SYS */

int relocate(void);


/*
**  File: FILES.C
*/

/*
** This is the format for a System File Table header.  SFT's are a linked
** list in which the header points to the next SFT, is followed by the
** number of FILES in this SFT, and ends with the FILES themselves, which
** are not important here.
*/

struct SFT_HEADER {
      struct SFT_HEADER (FAR *next);
      unsigned number;
};

int files(void);


#endif /* DOSFILES__H */
```

TEXT STATISTICS

2444 characters
66 lines

LEXICAL STATISTICS

28 comments [std-C]
0 comments [C++]
7 preprocessor instructions
0 constants [character]
2 constants [string]
1 constants [numeric]

SYMBOL TABLE

Boolean_T	15	19	20	21	27	34	
DOSFILES__H		7	8				
FAR	33+	59					
FILE	24	25	36				
PopDir	31						
PushDir	30						
SFT_HEADER		58	59				
TABLE_SIZE		42					
addpath	15						
chdrv	19						
copy	16						
dir	32						
dr	26						
drive	19	20	21	23	27	28	34
drive_type		26					
drvalid	20						
drvrdy	21						
dst	33						
favail	22						
fd	35						
fdupdate	35						
files	63						
fname	16						
fp	24	25	36				
fromDir	16						
fupdate	36						
getdcwd	23						
getdrv	18						
h	10						
isCDROMdrive		34					
isRamDsk	27						
isWprot	28						
iscons	24						
isdir	32						
isfopen	25						
mkdirs	29						
newdir	15	30					
next	59						
number	60						
pathname	29						
relocate	44						
src	33						
stdio	10						
toDir	17						
truename	33						

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FORMAT.C - Use DOS FORMAT to format a diskette
**
**  Original Copyright 1992 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdio.h>
#include <stdlib.h>
#include "snpdosys.h"

/*
**  format
**
**  Formats a specified floppy disk with optional switches.
**
**  Parameters: 1 - Drive letter ('A', 'B', ...) to format
**              2 - Formatting switches in FORMAT.COM format, e.g. "/4"
**              3 - Volume label
**
**  Returns: Success_ or Error_
*/

int format(char drive, char *switches, char *vlabel)
{
      char command[128], fname[13];
      FILE *tmpfile;

      tmpnam(fname);
      if (NULL == (tmpfile = fopen(fname, "w")))
            return Error_;                      /* Can't open temp file */
      fprintf(tmpfile, "\n%s\nN\n", vlabel);
      fclose(tmpfile);

      sprintf(command, "format %c: /V %s < %s > NUL", drive, switches, fname);

      system(command);

      remove(fname);

      return Success_;
}

#ifdef TEST

main()
{
      int retval = format((char)'a', "/4", "dummy_test");

      printf("format() returned %d\n", retval);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1458 characters
    60 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
     7 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    Error_          37
    FILE            33
    NULL            36
    Success_        47
    TEST            50
    command         32      41      43
    drive           30      41
    fclose          39
    fname           32      35      36      41      45
    fopen           36
    format          30      54
    fprintf         38
    h       14      15
    main            52
    printf          56
    remove          45
    retval          54      56
    sprintf         41
    stdio           14
    stdlib          15
    switches        30      41
    system          43
    tmpfile         33      36      38      39
    tmpnam          35
    vlabel          30      38
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 |    Header for DOSGETCH.ASM
   5 |    Donated to the public domain 96-11-12 by Tom Torfs (2:292/516)
   6 | */
   7 |
   8 | #ifndef DOSGETCH__H
   9 | #define DOSGETCH__H
  10 |
  11 | #include "extkword.h"
  12 |
  13 | extern int CDECL dosgetch(void);
  14 | extern int CDECL doswaitch(void);
  15 |
  16 | #define doswaitkey() if (doswaitch()==0) doswaitch()
  17 |
  18 | #endif /* DOSGETCH__H */
```

TEXT STATISTICS

```
   371 characters
    18 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    CDECL           13      14
    DOSGETCH__H              8       9
    dosgetch        13
    doswaitch       14      16+
    doswaitkey              16
```

```
/* +++Date last modified: 05-Jul-1997 */

/*

    cl /AL sortit.c

============================================================
   sort.c       7-31-91  Robert Mashlan

   This filter is almost compatible with the MS-DOS filter of the same name.

   This filter sorts each line of standard input, disregarding case,
   and sends it to standard output.

   optional parameters:  /R    Output in reverse order
                         /+n   Compare at column n, 1 based

   example usage:  sort < unsorted.txt > sorted.txt

   note: compile in a far data model for maximum capacity ( compact or large )

*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define MAXLINES 10000        /* maximum number of lines to sort */
#define MAXLINE  80           /* maximum line length             */

unsigned col     = 0;  /* column to start sort at ( zero based here ) */
int      reverse = 0;  /* reverse order flag                          */

/*
**  compare function for qsort
*/

int cmp( const void *a, const void *b)
{
      int result;
      const char *_a = *(const char **)a;
      const char *_b = *(const char **)b;

      /* compare at col if other than zero */

      if (col > 0)
      {
            if (strlen(_a) > col)
                  _a += col;
            else  _a = "";
            if (strlen(_b) > col)
                  _b += col;
            else  _b = "";
      }
      result = stricmp(_a,_b);
      return reverse ? -result : result;
}

int main(int argc, char *argv[])
{
      static char *lines[MAXLINES];
      int i, nlines=0, no_match;
      char buf[MAXLINE];

      /* scan for command line options */

      for(i=1;i<argc;i++)
      {
            if(!stricmp(argv[i],"/R"))  /* reverse order option */
                  reverse=1;
            else if (!strnicmp(argv[i],"/+",2))  /* column number option */
            {
                  int n = atoi(argv[i]+1)-1;

                  if (n < 0)
                        n = 0;
                  col = (unsigned)n;
            }
            else
            {
                  fputs("Invalid Parameter",stderr);
                  return 1;
            }
      }
      while(1)
      {
            /* read the line */

            if(fgets(buf,MAXLINE,stdin)==NULL)
                  break;

            /* Check for duplicates here */

            no_match = 1;
            for ( i=0; i< nlines; i++ )
            {
                  if (0 == (no_match = memicmp(buf, lines[i],
                        (unsigned int)12)))
                  {
                        break;
```

```
101 |                   }
102 |             }
103 |
104 |             if (no_match)                /* make a duplicate on the heap */
105 |             {
106 |                   if((lines[nlines++]=strdup(buf))==NULL)
107 |                   {
108 |                         fputs("SORT: Insufficient memory",stderr);
109 |                         return 2;
110 |                   }
111 |             }
112 |       }
113 |       /* sort the lot */
114 |
115 |       qsort((void *)lines,nlines,sizeof(char *),cmp);
116 |
117 |       /* display the sorted output */
118 |
119 |       for (i=0;i<nlines;i++)
120 |             fputs(lines[i],stdout); /* send it to standard output */
121 |       return 0;
122 | }
```

TEXT STATISTICS

```
  3280 characters
   122 lines
```

LEXICAL STATISTICS

```
    17 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     6 constants [string]
    22 constants [numeric]
```

SYMBOL TABLE

```
    MAXLINE          29      63      89
    MAXLINES         28      61
    NULL             89     106
    _a               41      48      49      50      55
    _b               42      51      52      53      55
    a        38      41
    argc             59      67
    argv             59      69      71      73
    atoi             73
    b        38      42
    buf              63      89      97     106
    cmp              38     115
    col              31      46      48      49      51      52      77
    fgets            89
    fputs            81     108     120
    h        24      25      26
    i        62      67+     69      71      73      95+     97     119+    120
    lines            61      97     106     115     120
    main             59
    memicmp          97
    n        73      75      76      77
    nlines           62      95     106     115     119
    no_match         62      94      97     104
    qsort           115
    result           40      55      56+
    reverse          32      56      70
    stderr           81     108
    stdin            89
    stdio            25
    stdlib           24
    stdout          120
    strdup          106
    stricmp          55      69
    string           26
    strlen           48      51
    strnicmp         71
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DOW.C
**
**  Public domain by VinhHao Nguyen, 03/94
**
**  Note: This function is redundant with the function of the same name
**        in DAYNUM.C. Which one to use?
**
**        If you're already using any functions in SCALDATE.C, DAYNUM.C,
**        or TODAY.C, ignore this file and use the other dow().
**
**        If you need a stand-alone DOW() function without using any of
**        the rest of the SCALDATE package, use this file.
*/

#include "scaldate.h"

#if ISO_CAL       /* monday == 0 */
 #define ADJ 5
#else             /* sunday == 0 */
 #define ADJ 6
#endif

unsigned DOW(unsigned y, unsigned m, unsigned d)
{
      if (m < 3)
      {
            m += 13;
            y--;
      }
      else  m++;
      return (d + 26 * m / 10 + y + y / 4 - y / 100 + y / 400 + ADJ) % 7;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      int Day;
      void usage(void);
      unsigned d, m, y, days[] = {31, 29, 31, 30, 31, 30,
                                  31, 31, 30, 31, 30, 31};
      char *day[2][7] = {
            {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"},
            {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"}
      };
      char *month[]  = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",};


      if (4 > argc)
            usage();
      y = atoi(argv[1]);
      m = atoi(argv[2]);
      d = atoi(argv[3]);
      if (!m || m > 12)
            usage();
      if (!d || d > days[m - 1])
            usage();
      if (y < 100)
            y += 1900;
      Day = DOW(y, m, d);
      printf("DOW(ISO_CAL=%d) returned %d, so %d %s %d is a %s\n",
            ISO_CAL, Day, d, month[m - 1], y, day[ADJ - 5][Day]);
      return 0;
}

void usage(void)
{
      puts("Usage: DOW yy[yy] mm dd");
      exit(-1);
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1995 characters
    79 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
    29 constants [string]
    36 constants [numeric]

SYMBOL TABLE

    ADJ             21      23      34      69
    DOW             26      67
    Day             44      67      69+
    ISO_CAL         20      69
    TEST            37
    argc            42      56
    argv            42      58      59      60
    atoi            58      59      60
    d       26      34      46      60      63+     67      69
    day             48      69
    days            46      63
    exit            76
    h       39      40
    m       26      28      30      33      34      46      59      61+     63      67      69
    main            42
    month           52      69
    printf          68
    puts            75
    stdio           39
    stdlib          40
    usage           45      57      62      64      73
    y       26      31      34+     46      58      65      66      67      69
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DRIVSRCH.C - public domain by Marty Connelly, Victoria, BC 1992
**
**  Modified by Bob Stout
**
**  Routine checks how many valid disk drives are available on machine,
**  both physical and logical drives
*/

/*
**  Includes drive letters assigned with DOS SUBST command
**
**  Networked drives are left as an exercise as I don't have access
**  to them to check.
**
**  The routine uses undocumented DOS interrupt 32H.
**
**  Compatible with MSC 5 and 6, ZTC++, BC++, other DOS compilers
**
**  DS:BX contains the address of the Disk Parameter Block (DPB) for a
**  requested drive. If the drive letter at offset 0 of the DPB doesn't
**  match the requested drive, then the drive has been SUBST'ed.
*/

#include <stdio.h>
#include <dos.h>
#include "extkword.h"
#include "mk_fp.h"

main()
{
      int i;
      int unsigned result;
      int drivestatus[26];
      unsigned char FAR *DPB;
      union REGS regs;
      struct SREGS sregs;


      /* routine checks for all valid drive possibilities from A to Z  */

      /*
      **    if removable media drive ie. floppy drive A: has a latch door
      **    open you will get "Abort Retry" panic message
      */

      for (i = 0; i < 26; i++)
      {
            /* drive number (0=default, 1=A, 2=B,etc.)*/

            regs.h.dl = (unsigned char)(i + 1);
            segread(&sregs);

            regs.h.ah=0x32;         /* DOS interrupt 32H */
                                    /* was undocumented for DOS release 3.2 */

            intdosx(&regs,&regs, &sregs);

            result=regs.h.al;
            DPB = MK_FP(sregs.ds, regs.x.bx);

            /*
            **  result =0  then valid drive
            **         =255 or ff hex then invalid or non-existent drive
            */

            if (0 == result && *DPB != (unsigned char)i)
                  drivestatus[i] = 1;
            else  drivestatus[i]=result;
      }

      for (i = 0; i < 26; i = i + 2)
      {
            printf("drive %c: status code =%3d drive %c: status code =%3d\n",
                  'A' + i,drivestatus[i],'B' + i,drivestatus[i+1]);
      }
      return 0;
}
```

```
TEXT STATISTICS

  2241 characters
    80 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     2 constants [character]
     3 constants [string]
    12 constants [numeric]

SYMBOL TABLE

    DPB             37      62      69
    FAR             37
    MK_FP           62
    REGS            38
    SREGS           39
    ah              56
    al              61
    bx              62
    dl              53
    dos             28
    drivestatus             36      70      71      77+
    ds              62
    h       27      28      53      56      61
    i       34      49+     53      69      70      71      74+     77+
    intdosx         59
    main            32
    printf          76
    regs            38      53      56      59+     61      62
    result          35      61      69      71
    segread         54
    sregs           39      54      59      62
    stdio           27
    x       62
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  DRVALID.C - validate disk drives
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Uses ABSDISKC.C and ABSDISK.ASM in SNIPPETS.
*/

#include <dos.h>
#include <stdlib.h>
#include "dosfiles.h"
#include "snpdskio.h"

/*
**  getdrv()
**
**  Just as getcwd() returns the default directory, getdrv() returns
**  the current drive.
**
**  Arguments: None.
**
**  Returns:   Current drive (0 = A:, 1 = B:, etc.)
**
**  Side effects: none
*/

int getdrv(void)
{
      union REGS regs;

      regs.h.ah = 0x19;
      intdos(&regs, &regs);
      return (regs.h.al);
}

/*
**  chdrv()
**
**  Like chdir(), except changes drives rather than directories.
**
**  Arguments: 1 - target drive (0 = A:, 1 = B:, etc.)
**
**  Returns: Success_ or Error_
**
**  Side effects: none
*/

Boolean_T chdrv(int drive)
{
      union REGS regs;

      regs.h.ah = 0x0e;
      regs.h.dl = (char)drive;
      intdos(&regs, &regs);
      if (drive != getdrv())
            return Error_;
      else  return Success_;
}

/*
**  drvalid()
**
**  Verifies whether a logical disk drive is available without
**  triggering the DOS critical error handler.
**
**  Arguments: 1 - target drive (0 = A;, 1 = B:, etc.)
**
**  Returns:   True_  - drive is valid
**             False_ - drive is invalid
**
**  Side effects: none
*/

Boolean_T drvalid(int drive)
{
      int original, result;

      original = getdrv();
      result   = (Success_ == chdrv(drive));
      chdrv(original);
      return result;
}

/*
**  drvrdy()
**
**  Checks whether a drive with removable media is ready.
**
**  Arguments: 1 - target drive (0 = A;, 1 = B:, etc.)
**
**  Returns:   True_  - drive is ready
**             False_ - drive is not ready
**             Error_ - other read error
```

```
**
**  Side effects: none
*/

Boolean_T drvrdy(int drive)
{
      int status;
      char buf[2048];         /* nice & roomy   */

      status = AbsDiskRead(drive, 1, 0, buf);
      if (0 == status)
            return True_;
      status &= 0xff;
      if (2 == status)
            return False_;
      else  return Error_;
}

#ifdef TEST

#include <stdio.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
      int drive;

      if (2 > argc)
      {
            puts("Usage: DRVALID drive[:]");
            return EXIT_FAILURE;
      }
      drive = toupper(*argv[1]);
      if (!isalpha(drive))
      {
            puts("Error: Invalid drive name");
            return EXIT_FAILURE;
      }
      printf("Drive %c: is %svalid\n", drive,
            drvalid(drive - 'A') ? "" : "not ");
      if (2 < _osmajor)
      {
            union REGS regs;

            regs.x.ax = 0x4408;
            regs.h.bl = (unsigned char)(drive - '@');
            intdos(&regs, &regs);
            printf("ioctl returned Cflag=%s\n",
                  regs.x.cflag ? "TRUE" : "FALSE");
            printf("ioctl returned AX=0x%X\n", regs.x.ax);
            printf("Drive %c is%s removable\n", drive,
                  regs.x.ax ? " not" : "");
            if (0 == regs.x.ax)
            {
                  printf("Drive %c is %sready\n", drive,
                        drvrdy(drive - 'A') ? "" : "not ");
            }
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  3667 characters
   162 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     3 constants [character]
    17 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    AbsDiskRead            110
    Boolean_T       55      81     105
    EXIT_FAILURE           131     137
    EXIT_SUCCESS           159
    Error_          63     116
    False_         115
    REGS            36      57     143
    Success_        64      86
    TEST           119
    True_          112
    _osmajor       141
    ah              38      59
    al              40
    argc           124     128
    argv           124     133
    ax             145     150     152     153
    bl             146
    buf            108     110
    cflag          149
    chdrv           55      86      87
    ctype          122
    dl              60
    dos             16
    drive           55      60      62      81      86     105     110     126     133     134     139
           140     146     151     155     156
    drvalid         81     140
    drvrdy         105     156
    getdrv          34      62      85
    h       16      17      38      40      59      60     121     122     146
    intdos          39      61     147
    isalpha        134
    main           124
    original        83      85      87
    printf         139     148     150     151     155
    puts           130     136
    regs            36      38      39+     40      57      59      60      61+    143     145     146
           147+    149     150     152     153
    result          83      86      88
    status         107     110     111     113     114
    stdio          121
    stdlib          17
    toupper        133
    x       145    149     150     152     153
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DRVS.C - public domain by David Gersic, DeKalb, Il 1993
  5 | **
  6 | **  Routine checks how many valid disk drives are available on machine,
  7 | **  both physical and logical drives.
  8 | **
  9 | **  Includes drive letters assigned with DOS SUBST command and network
 10 | **  drives for Novell Netware (and probably other networks).
 11 | **
 12 | **  Compiled Under MSC 6 LARGE memory Model
 13 | **  Should be compatible with other DOS compilers
 14 | **
 15 | */
 16 |
 17 | #include <stdio.h>
 18 | #include <dos.h>
 19 | #include <stdlib.h>
 20 |
 21 | main()
 22 | {
 23 |       union REGS in, out;
 24 |       int i;
 25 |
 26 |       /* Novell's shell TSRs allow up to 32 drive 'letters' to be created */
 27 |
 28 |       char drives[]={' ','a','b','c','d','e','f','g','h','i','j',
 29 |                       'k','l','m','n','o','p','q','r','s','t','u',
 30 |                       'v','w','x','y','z','[','\\',']','^','_','`'};
 31 |
 32 |       in.x.ax=0x4409;   /* IOCTL function - Check if block device remote  */
 33 |       for(i = 1; i < 32; i++)
 34 |       {
 35 |             in.h.bl=(unsigned char)i;     /* 1==a:, 2==b:, etc.           */
 36 |             intdos(&in,&out);
 37 |             if(!out.x.cflag)       /* carry flag set on error            */
 38 |             {                      /* bit 15 == subst, bit 12 == 'remote'*/
 39 |                   printf("drive %c: is %s\n",
 40 |                         drives[i],out.x.dx & (1<<15) ? "subst" :
 41 |                         out.x.dx & (1<<12) ? "network" : "local");
 42 |             }
 43 |       }
 44 |       return(0);
 45 | }
```

TEXT STATISTICS

```
  1482 characters
    45 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
    33 constants [character]
     4 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    REGS            23
    ax              32
    bl              35
    cflag           37
    dos             18
    drives          28      40
    dx              40      41
    h       17      18      19      35
    i       24      33+     35      40
    in              23      32      35      36
    intdos          36
    main            21
    out             23      36      37      40      41
    printf          39
    stdio           17
    stdlib          19
    x       32      37      40      41
```

```
/* +++Date last modified: 05-Jul-1997 */

/* public domain TSR clock code. By Michelangelo Jones, 1:141/575 or
// 1:1/124. Very little support available; this is a quick hack, not
// an example of the best way to write a clock TSR! Use at own risk.
// Runs under TC++/BC++. Your mileage may vary.
*/

#ifndef __HUGE__
#error "Must be compiled using HUGE model-- for now"
#endif

#include <dos.h>

extern unsigned _heaplen = 0;
extern unsigned _stklen = 512;

static char * screenbase =
           (char *)                 MK_FP (0xb800, 0); /* change for mono */
static const long int * volatile ticks =                 /* to 0xb000, 0 */
           (long int * volatile)    MK_FP (0, 0x046c);

int calls, lastsec, lastmin, lasthr;
const double tps = 18.20648;          /* found by experimentation! */

void interrupt (*oldhandler)(void);

void displayclock (void)
{
    char *moveinto = screenbase + 300;
    char *initwith = "      :     :       ";

/* NOTE: This initializer only works because the attribute I want for the
//       clock HAPPENS to be the same as the ASCII value for the SPACE char!
//       Modify every alternate character if you want some other attribute.
*/

    while (*initwith)
        *moveinto++ = *initwith++;
    lastsec = -1;
    lastmin = -1;
    lasthr  = -1;
    calls   = 20;
}

void interrupt clockproc(void)
{
    static long seconds;

    if (calls < 17)
        calls++;
    else
    {
        seconds = (long) ((double) *ticks / tps);
        if (screenbase[301] != ' ') /* if the attribute has changed, */
            displayclock();         /* the screen scrolled, so update. */
        if (seconds % 60 != lastsec)
        {
            lastsec = seconds % 60;
            calls = 0;
            screenbase[314] = (char) (lastsec/10) + 48;
            screenbase[316] = (char) (lastsec%10) + 48;
            if ((! lastsec) || (lastmin < 0))
            {
                lastmin = (seconds % 3600) / 60;
                screenbase[308] = (char) (lastmin/10) + 48;
                screenbase[310] = (char) (lastmin%10) + 48;
                if ((! lastmin) || (lasthr < 0))
                {
                    lasthr = ((seconds % 86400L) / 3600L);
                    screenbase[302] = (char) (lasthr/10) + 48;
                    screenbase[304] = (char) (lasthr%10) + 48;
                }
            }
        }
    }
    oldhandler ();
}

main()
{
    oldhandler = getvect (0x1c);
    displayclock();
    setvect (0x1c, clockproc);
    keep (0, (_SS + (_SP/16) - _psp));
    return 0;
}
```

TEXT STATISTICS

```
  2627 characters
    87 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     1 constants [character]
     2 constants [string]
    46 constants [numeric]
```

SYMBOL TABLE

```
    MK_FP           19      21
    _SP             85
    _SS             85
    __HUGE__         9
    _heaplen        15
    _psp            85
    _stklen         16
    calls           23      43      50      51      60
    clockproc       46      84
    displayclock            28      56      83
    dos             13
    getvect         82
    h       13
    initwith        31      38      39
    interrupt       26      46
    keep            85
    lasthr          23      42      68      70      71      72
    lastmin         23      41      63      65      66      67      68
    lastsec         23      40      57      59      61      62      63
    main            80
    moveinto        30      39
    oldhandler              26      77      82
    screenbase              18      30      55      61      62      66      67      71      72
    seconds         48      54      57      59      65      70
    setvect         84
    ticks           20      54
    tps             24      54
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Compiler I/O benchmarks
**  public domain by Dave Knapp & Bob Stout
*/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include "errors.h"           /* For cant()                              */
#include "sniptype.h"         /* For DWORD                               */
#include "extkword.h"         /* For FAR                                 */
#include "scrnmacs.h"         /* For portable screen I/O                 */
#include "dvidport.h"         /* For portable direct video I/O           */

#ifndef ITERATIONS
 #define ITERATIONS 5000
#endif

#ifdef M_I86      /* Identifier for MSC, QC, Watcom, or ZTC */
 #ifndef __ZTC__
  #include <graph.h>
  #ifndef __WATCOMC__
   #ifdef _MSC_VER
    #define LOGFILE "dspdtst.msc"
   #else
    #define LOGFILE "dspdtst.qc"
   #endif
  #else
   #define LOGFILE "dspdtst.wc"
  #endif    /* not Watcom   */
 #else      /* if ZTC       */
  #include <disp.h>
  #ifdef __SC__
   #define LOGFILE "dspdtst.sc"
  #else
   #define LOGFILE "dspdtst.ztc"
  #endif
 #endif     /* if ZTC       */
#else       /* must be Borland or Mix     */
 #ifdef __TURBOC__
  #ifdef __BORLANDC__
   #define LOGFILE "dspdtst.bc"
  #else
   #define LOGFILE "dspdtst.tc"
  #endif
 #endif
 #ifdef __POWERC
  #define LOGFILE "dspdtst.pc"
 #endif
#endif

#ifndef LOGFILE
 #error Unrecognized compiler
#endif

DWORD FAR *bios_time = (DWORD FAR *)(0x0040006cL);
DWORD time1, time2, time3, time4, time5, time6;

main()
{
        int i;
        FILE *log = stdout, *nulfile;

        ClrScrn(BG_(BLACK_)+GRAY_);
        nulfile = cant("NUL", "w");
        time1 = *bios_time;
        for(i = 1; i < ITERATIONS; i++)
        {
                gotoxy(10,5);
                puts("puts      test.");
                puts("this is the second line.\n");
        }
        time1 = *bios_time - time1;
        time2 = *bios_time;
        for(i = 1; i < ITERATIONS; i++)
        {
                gotoxy(10,8);
                printf("printf    test.\n");
                printf("this is the second line.\n");
        }
        time2 = *bios_time - time2;
        dvid_open();
        time3 = *bios_time;
        for(i = 1; i < ITERATIONS; i++)
        {
                gotoxy(10,11);
                cputs("cputs     test.\r\n");
                cputs("this is the second line.\r\n");
        }
        time3 = *bios_time - time3;
        time4 = *bios_time;
        for(i = 1; i < ITERATIONS; i++)
        {
                gotoxy(10,14);
                cprintf("cprintf   test.\r\n");
                cprintf("this is the second line.\r\n");
        }
        time4 = *bios_time - time4;
```

```
 101 |         dvid_close();
 102 |         time5 = *bios_time;
 103 |         for(i = 1; i < ITERATIONS; i++)
 104 |         {
 105 |                 fputs("fputs    test.\n", nulfile);
 106 |                 fputs("this is the second line.\n", nulfile);
 107 |         }
 108 |         time5 = *bios_time - time5;
 109 |         time6 = *bios_time;
 110 |         for(i = 1; i < ITERATIONS; i++)
 111 |         {
 112 |                 fprintf(nulfile, "fprintf   test.\n");
 113 |                 fprintf(nulfile, "this is the second line.\n");
 114 |         }
 115 |         time6 = *bios_time - time6;
 116 |
 117 |         log = cant(LOGFILE, "w");
 118 |         fprintf(log, "Times for %d iterations:\n\n", ITERATIONS);
 119 |         fprintf(log, "puts     %10.3f seconds\n", (double)time1 * .054945);
 120 |         fprintf(log, "printf   %10.3f seconds\n", (double)time2 * .054945);
 121 |         fprintf(log, "cputs    %10.3f seconds\n", (double)time3 * .054945);
 122 |         fprintf(log, "cprintf  %10.3f seconds\n", (double)time4 * .054945);
 123 |         fprintf(log, "fputs    %10.3f seconds\n", (double)time5 * .054945);
 124 |         fprintf(log, "fprintf  %10.3f seconds\n", (double)time6 * .054945);
 125 |         fclose(log);
 126 |         printf("\nDone - results are in %s\n", LOGFILE);
 127 |         return 0;
 128 | }
```

TEXT STATISTICS

```
  3935 characters
   128 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    46 preprocessor instructions
     0 constants [character]
    36 constants [string]
    23 constants [numeric]
```

SYMBOL TABLE

```
    BG_             66
    BLACK_          66
    ClrScrn         66
    DWORD           58+     59
    FAR             58+
    FILE            64
    GRAY_           66
    ITERATIONS              17      18      69      77      86      94     103     110     118
    LOGFILE         26      28      31      36      38      44      46      50      54     117     126
    M_I86           21
    Unrecognized            55
    _MSC_VER        25
    __BORLANDC__            43
    __POWERC        49
    __SC__          35
    __TURBOC__              42
    __WATCOMC__             24
    __ZTC__         22
    bios_time       58      68      75      76      83      85      92      93     100     102     108
           109     115
    cant            67     117
    compiler        55
    conio            9
    cprintf         97      98
    cputs           89      90
    disp            34
    dos             10
    dvid_close              101
    dvid_open       84
    fclose         125
    fprintf        112     113     118     119     120     121     122     123     124
    fputs          105     106
    gotoxy          71      79      88      96
    graph           23
    h        8       9      10      23      34
    i       63      69+     77+     86+     94+    103+    110+
    log             64     117     118     119     120     121     122     123     124     125
    main            61
    nulfile         64      67     105     106     112     113
    printf          80      81     126
    puts            72      73
    stdio            8
    stdout          64
    time1           59      68      75+    119
    time2           59      76      83+    120
    time3           59      85      92+    121
    time4           59      93     100+    122
    time5           59     102     108+    123
    time6           59     109     115+    124
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
$Header: c:/bnxl/rcs/dtotp6.c 1.2 1995/05/01 00:30:58 bnelson Exp $

$Log: dtotp6.c $
Revision 1.2  1995/05/01 00:30:58  bnelson

- Added test driver and Thad Smith's original function
  to convert a TP real to double

- Checks out OK with lint.

- Tested on GNU C 2.6.3 (little endian Intel) after adding PAK to
  real struct.

Revision 1.1  1995/04/30 23:54:56  bnelson

Written by Bob Nelson of Dallas, TX, USA (bnelson@netcom.com)

Original tp6_to_double() written by Thad Smith III of Boulder, CO, and
  released to the public domain in SNIPPETS

- Initial release -- converts C double value into the bit pattern used
  by a Turbo Pascal 6-byte real. Uses the "real" struct written by Thad
  Smith for ease of assignment to members.

- Tested on BC++ 3.1.

- This source and associated include are contributed to the Public Domain.
*/


#include <math.h>
#include "dtotp6.h"


#define DBL_BIAS            0x3FE
#define REAL_BIAS           0x80
#define TP_REAL_BIAS        (DBL_BIAS - REAL_BIAS)    /* 0x37E */


tp_real_t double_to_tp6(double x)
{
      unsigned int *wp;
      tp_real_t r;

      if(x == 0.0)
      {
            r.v3 = r.v2 = r.v1 = r.be = r.s = 0;
            return r;
      }

      wp = (void *)&x;            /* Break down double into words */

      r.s  = wp[3] >> 15;         /* High bit set for sign */

      /* ------------------------------------------------------------------
      ** Grab biased exponent -- exclude sign and shift out the MSB
      ** mantissa bits.
      */

      r.be = (unsigned char)(((wp[3] & 0x7FFF) >> 4) - TP_REAL_BIAS);

      /* ------------------------------------------------------------------
      ** Now...just assign the mantissa after shifting the bits to conform
      ** with the layout for the TP 6-byte real.
      */

      r.v3 = ((wp[3] & 0x0F) << 3) | (wp[2] >> 13);
      r.v2 = (wp[2] << 3) | (wp[1] >> 13);
      r.v1 = (wp[1] << 3) | (wp[0] >> 13);

      return r;
}


/* ------------------------------------------------------------------
** Slightly adapted version of Thad Smith's function from TP6TOD.C
** from Snippets. (Uses TP real struct parameter and no memcpy).
*/


double tp6_to_double(tp_real_t r)
{
      if (r.be == 0)
            return 0.0;

      return ((((128 + r.v3) * 65536.0) + r.v2) * 65536.0 + r.v1) *
            ldexp((r.s ? -1.0 : 1.0), r.be - (129 + 39));
}

#if defined (TEST)

#include <stdio.h>
#include <stdlib.h>


int main(int argc, char **argv)
```

```
 101 | {
 102 |       double x, y;
 103 |       tp_real_t r;
 104 |
 105 |       if(argc > 1)
 106 |             x = strtod(argv[1], NULL);
 107 |       else  x = 19999.99;
 108 |
 109 |       r = double_to_tp6(x);
 110 |       y = tp6_to_double(r);
 111 |
 112 |       printf("input double value: %.2f, converted double value: %.2f\n",
 113 |             x, y);
 114 |
 115 |       return 0;
 116 | }
 117 |
 118 | #endif    /* TEST */
```

TEXT STATISTICS

```
  2947 characters
   118 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     2 constants [string]
    35 constants [numeric]
```

SYMBOL TABLE

```
    DBL_BIAS        38      40
    NULL           106
    REAL_BIAS       39      40
    TEST            93
    TP_REAL_BIAS            40      63
    argc           100     105
    argv           100     106
    be              50      63      86      90
    defined         93
    double_to_tp6           43     109
    h       34      95      96
    ldexp           90
    main           100
    math            34
    printf         112
    r       46      50+     51      56      63      70      71      72      74      84      86      89+
            90+    103     109     110
    s       50      56      90
    stdio           95
    stdlib          96
    strtod         106
    tp6_to_double           84     110
    tp_real_t       43      46      84     103
    v1              50      72      89
    v2              50      71      89
    v3              50      70      89
    wp              45      54      56      63      70+     71+     72+
    x       43      48      54     102     106     107     109     113
    y      102     110     113
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | $Header: c:/bnxl/rcs/dtotp6.h 1.1 1995/05/01 00:04:08 bnelson Exp $
  5 |
  6 | $Log: dtotp6.h $
  7 | Revision 1.1  1995/05/01 00:04:08  bnelson
  8 |
  9 | - Include file companion to dtotp6.c, see notes contained there.
 10 | */
 11 |
 12 |
 13 | #include "dirport.h"
 14 |
 15 |
 16 | #ifndef D2TOTP6_H_
 17 | #define     D2TOTP6_H_
 18 |
 19 | #ifdef __TURBOC__
 20 | #pragma     option -a-       /* Force byte alignment in struct */
 21 | #endif
 22 |
 23 | #ifdef __GNUC__
 24 | #define PAK        __attribute__((packed))
 25 | #else
 26 | #define PAK
 27 | #endif
 28 |
 29 | #ifdef MONOSPACE_6           /* Just to be safe... */
 30 |  #define     double_to_tp6    DBL2TP
 31 |  #define     tp6_to_double    TP2DBL
 32 | #endif
 33 |
 34 | typedef struct {
 35 |     unsigned char be   PAK;     /* biased exponent */
 36 |     unsigned int  v1   PAK;     /* lower 16 bits of mantissa */
 37 |     unsigned int  v2   PAK;     /* next  16 bits of mantissa */
 38 |     unsigned int  v3:7 PAK;     /* upper  7 bits of mantissa */
 39 |     unsigned int  s :1 PAK;     /* sign bit */
 40 | } tp_real_t;
 41 |
 42 | extern tp_real_t  double_to_tp6(double x);
 43 | extern double     tp6_to_double(tp_real_t r);
 44 |
 45 | #endif    /* D2TOTP6_H_ */
```

TEXT STATISTICS

```
  1110 characters
    45 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    D2TOTP6_H_              16      17
    DBL2TP          30
    MONOSPACE_6             29
    PAK             24      26      35      36      37      38      39
    TP2DBL          31
    __GNUC__        23
    __TURBOC__              19
    __attribute__           24
    a       20
    be              35
    double_to_tp6           30      42
    option          20
    packed          24
    r       43
    s       39
    tp6_to_double           31      43
    tp_real_t       40      42      43
    v1              36
    v2              37
    v3              38
    x       42
```

```
/* +++Date last modified: 05-Jul-1997 */

/*----------------------------[ dvideo.c ]----------------------------*/
/*                     Hardware Interface Routines                     */
/*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*/
/* This code is a subset of a library copyrighted by Jeff Dunlop.      */
/* License is hereby granted for unrestricted use.                     */
/*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*/
/* dv_info: get video and version info about DesqView                  */
/* get_rows: determine number of rows on screen                        */
/* get_cols: determine number of columns on screen                     */
/* get_vidpage: determine the current text mode video page             */
/* get_vidbase: determine the base of video ram                        */
/* is_dv: determine if DesqView is loaded                              */
/* is_egavga: determine if monitor is ega/vga                          */
/*---------------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*----------------------[ include files ]----------------------*/
/*-------------------------------------------------------------*/

#include <dos.h>
#include "dvideo.h"
#include "mk_fp.h"

/*-----------------------[ get_vidpage ]-----------------------*/
/* Determine the current text mode video page                  */
/*-------------------------------------------------------------*/
/* local:                                                      */
/*      regs = register storage buffer                         */
/* return:                                                     */
/*      video page number as determined by bios call           */
/*-------------------------------------------------------------*/

unsigned char get_vidpage(void)
{
    union REGS regs;

    regs.h.ah = 0x0f;
    int86(0x10, &regs, &regs);
    return regs.h.bh;
}

/*---------------------------[ is_dv ]---------------------------*/
/*             Determine whether DesqView is active            */
/*-------------------------------------------------------------*/

int is_dv(void)
{
    union REGS regs;

    regs.h.ah = 0x2b;
    regs.x.cx = 0x4445;         /* 'DE' */
    regs.x.dx = 0x5351;         /* 'SQ' */
    regs.h.al = 1;              /* get version */
    int86(0x21, &regs, &regs);

    return regs.h.al != 0xff;
}

/*-------------------------[ dv_info ]-------------------------*/
/*      Return screen and version info about DesqView          */
/*-------------------------------------------------------------*/
/* return:                                                     */
/*      -1 on error                                            */
/*-------------------------------------------------------------*/

int get_dvinfo(DV_INFO *dv_info)
{
    union REGS regs;

    regs.h.ah = 0x2b;
    regs.x.cx = 0x4445;         /* 'DE' */
    regs.x.dx = 0x5351;         /* 'SQ' */
    regs.h.al = 1;              /* get version */
    int86(0x21, &regs, &regs);

    if (regs.h.al == 0xff)
        return -1;
    dv_info->ver_major = regs.h.bh;
    dv_info->ver_minor = regs.h.bl;
    regs.h.al = 4;              /* get screen info */
    int86(0x21, &regs, &regs);

    if (regs.h.al == 0xff)
        return -1;
    dv_info->regen_buf = regs.x.dx;
    dv_info->win_rows = regs.h.bh;
    dv_info->win_cols = regs.h.bl;
    return 0;
}

/*-----------------------[ get_vidbase ]-----------------------*/
/* Determine the base of video ram                             */
/*-------------------------------------------------------------*/
/* local:                                                      */
```

```
101 | /*      regs = register union for ISR                           */
102 | /* return:                                                      */
103 | /*      the current text base segment                           */
104 | /*--------------------------------------------------------------*/
105 |
106 | unsigned get_vidbase(void)
107 | {
108 |     union REGS regs;
109 |     DV_INFO dv_info;
110 |
111 |     if (is_dv() && get_dvinfo(&dv_info) != -1)
112 |         return dv_info.regen_buf;
113 |     else
114 |     {
115 |         regs.h.ah = 0xf;
116 |         int86(0x10, &regs, &regs);
117 |
118 |         if (regs.h.al == 7)
119 |             return 0xb000;
120 |         else
121 |             return 0xb800;
122 |     }
123 | }
124 |
125 | /*-------------------------[ get_rows ]-------------------------*/
126 | /*  Determine the number of rows in current text mode screen    */
127 | /*--------------------------------------------------------------*/
128 |
129 | int get_rows(void)
130 | {
131 |     DV_INFO dv_info;
132 |     char far *p = MK_FP(0x40, 0x84);
133 |
134 |     if (is_dv() && get_dvinfo(&dv_info) != -1)
135 |         return dv_info.win_rows;
136 |     else
137 |         return *p + is_egavga();
138 | }
139 |
140 | /*-------------------------[ get_cols ]-------------------------*/
141 | /*  Determine the number of columns in current text screen      */
142 | /*--------------------------------------------------------------*/
143 |
144 | int get_cols(void)
145 | {
146 |     DV_INFO dv_info;
147 |     int far *p = MK_FP(0x40, 0x4a);
148 |
149 |     if (is_dv() && get_dvinfo(&dv_info) != -1)
150 |         return dv_info.win_cols;
151 |     else
152 |         return *p;
153 | }
154 |
155 | /*-------------------------[ is_egavga ]------------------------*/
156 | /*      Determine whether the current text mode is ega/vga      */
157 | /*--------------------------------------------------------------*/
158 |
159 | int is_egavga(void)
160 | {
161 |     union REGS regs;
162 |
163 |     regs.h.ah = 0x1a;
164 |     regs.h.al = 0;
165 |
166 |     int86(0x10, &regs, &regs);
167 |
168 |     return regs.h.al == 0x1a;
169 | }
```

TEXT STATISTICS

```
  5837 characters
   169 lines
```

LEXICAL STATISTICS

```
    61 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     2 constants [string]
    36 constants [numeric]
```

SYMBOL TABLE

```
    DV_INFO          72     109     131     146
    MK_FP           132     147
    REGS             41      54      74     108     161
    ah               43      56      76     115     163
    al               59      62      79      82      86      89     118     164     168
    bh               45      84      92
    bl               85      93
    cx               57      77
    dos              26
    dv_info          72      84      85      91      92      93     109     111     112     131     134
           135     146     149     150
    dx               58      78      91
    far             132     147
    get_cols        144
    get_dvinfo               72     111     134     149
    get_rows        129
    get_vidbase             106
    get_vidpage              39
    h        26      43      45      56      59      62      76      79      82      84      85      86
             89      92      93     115     118     163     164     168
    int86            44      60      80      87     116     166
    is_dv            52     111     134     149
    is_egavga       137     159
    p       132     137     147     152
    regen_buf        91     112
    regs             41      43      44+     45      54      56      57      58      59      60+     62
            74      76      77      78      79      80+     82      84      85      86      87+     89
            91      92      93     108     115     116+    118     161     163     164     166+    168
    ver_major        84
    ver_minor        85
    win_cols         93     150
    win_rows         92     135
    x        57      58      77      78      91
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*----------------------------[ dvideo.h ]-----------------------------*/
   4 | /*                     Hardware Interface Routines                      */
   5 | /*----------------------------------------------------------------------*/
   6 |
   7 | #ifndef DVIDEO__H
   8 | #define DVIDEO__H
   9 |
  10 | typedef struct
  11 | {
  12 |     int ver_major;
  13 |     int ver_minor;
  14 |     unsigned regen_buf;
  15 |     int win_rows;
  16 |     int win_cols;
  17 | } DV_INFO;
  18 |
  19 | unsigned char     get_vidpage(void);
  20 | int               is_dv(void);
  21 | int               get_dvinfo(DV_INFO *dv_info);
  22 | unsigned          get_vidbase(void);
  23 | int               get_rows(void);
  24 | int               get_cols(void);
  25 | int               is_egavga(void);
  26 |
  27 | #endif /* DVIDEO__H */
```

TEXT STATISTICS

```
   736 characters
    27 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DVIDEO__H        7       8
    DV_INFO         17      21
    dv_info         21
    get_cols        24
    get_dvinfo              21
    get_rows        23
    get_vidbase             22
    get_vidpage             19
    is_dv           20
    is_egavga       25
    regen_buf       14
    ver_major       12
    ver_minor       13
    win_cols        16
    win_rows        15
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DVIDPORT.C - Direct video portability functions
  5 | */
  6 |
  7 | #ifdef __ZTC__                          /* Only used with SC/ZTC      */
  8 |
  9 | #include <disp.h>
 10 | #include "scrnmacs.h"
 11 |
 12 | void gotoxy(int col, int row)
 13 | {
 14 |       if (disp_inited)
 15 |             disp_move(row, col);
 16 |       else  GotoXY(col, row);
 17 | }
 18 |
 19 | #endif /* __ZTC__ */
```

TEXT STATISTICS

```
   379 characters
    19 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    GotoXY          16
    __ZTC__          7
    col             12      15      16
    disp             9
    disp_inited             14
    disp_move       15
    gotoxy          12
    h        9
    row             12      15      16
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  DVIDPORT.H - Direct video portability functions for PC compilers
  5 | **
  6 | **  For ease of use, makes everything look like Borland, but adds dvid_open()
  7 | **  and dvid_close() for Symantec & Zortech
  8 | **
  9 | **  public domain by Dave Knapp & Bob Stout
 10 | */
 11 |
 12 | #ifndef DVIDPORT__H
 13 | #define DVIDPORT__H
 14 |
 15 | #include <stdio.h>
 16 | #include <conio.h>
 17 | #include <dos.h>
 18 |
 19 | #ifdef M_I86      /* Identifier for MSC, QC, Watcom, or SC/ZTC    */
 20 |  #ifndef __ZTC__
 21 |   #include <graph.h>
 22 |   #define dvid_open()
 23 |   #define dvid_close()
 24 |   #define cputs(s) _outtext((char _far *)(s))
 25 |   #define gotoxy(col,row) _settextposition(row,col)
 26 |  #else   /* if SC/ZTC   */
 27 |   #include <disp.h>
 28 |   void gotoxy(int, int);
 29 |   #define dvid_open  disp_open
 30 |   #define dvid_close  disp_close
 31 |   #define cputs(s) disp_puts(s)
 32 |   #define cprintf(s) disp_printf(s)
 33 |  #endif  /* if ZTC      */
 34 | #else
 35 |  #define dvid_open()
 36 |  #define dvid_close()
 37 |  #if defined(__POWERC)
 38 |   #define gotoxy(col,row) poscurs(row,col)
 39 |  #endif
 40 | #endif
 41 |
 42 | #endif /* DVIDPORT__H */
```

TEXT STATISTICS

```
  1066 characters
    42 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    27 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DVIDPORT__H             12      13
    M_I86           19
    __POWERC        37
    __ZTC__         20
    _far            24
    _outtext        24
    _settextposition        25
    col             25+     38+
    conio           16
    cprintf         32
    cputs           24      31
    defined         37
    disp            27
    disp_close              30
    disp_open       29
    disp_printf             32
    disp_puts       31
    dos             17
    dvid_close              23      30      36
    dvid_open       22      29      35
    gotoxy          25      28      38
    graph           21
    h       15      16      17      21      27
    poscurs         38
    row             25+     38+
    s       24+     31+     32+
    stdio           15
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  EASTER.C - Determine the date of Easter for any given year
 5 | **
 6 | **  public domain by Ed Bernal
 7 | */
 8 |
 9 | #include <stdlib.h>
10 | #include "datetime.h"
11 |
12 |
13 | void easter(int year,int *easter_month, int *easter_day)
14 | {
15 |       int a,b,c,e,g,h,i,k,u,x,z;
16 |       div_t f;
17 |
18 |       /*
19 |       **  Gauss' famous algorithm (I don't know how or why it works,
20 |       **  so there's no commenting)
21 |       */
22 |
23 |       a = year % 19;
24 |       f = div(year,100);
25 |       b = f.quot;
26 |       c = f.rem;
27 |       f = div(b,4);
28 |       z = f.quot;
29 |       e = f.rem;
30 |       f = div((8*b + 13),25);
31 |       g = f.quot;
32 |       f = div((19*a + b - z - g + 15),30);
33 |       h = f.rem;
34 |       f = div((a + 11*h),319);
35 |       u = f.quot;
36 |       f = div(c,4);
37 |       i = f.quot;
38 |       k = f.rem;
39 |       f = div((2*e + 2*i - k - h + u + 32),7);
40 |       x = f.rem;
41 |       f = div((h-u+x+90),25);
42 |       *easter_month = f.quot;
43 |       f = div((h-u+x + *easter_month +19),32);
44 |       *easter_day = f.rem;
45 | }
46 |
47 | #ifdef TEST
48 |
49 | #include <stdio.h>
50 |
51 | main(int argc, char *argv[])
52 | {
53 |       int yr, mo, dy;
54 |
55 |       while (--argc)
56 |       {
57 |             yr = atoi(*++argv);
58 |             if (100 > yr)
59 |                   yr += 1900;
60 |             easter(yr, &mo, &dy);
61 |             printf("Easter in %d on %d/%d\n", yr, mo, dy);
62 |       }
63 |       return 0;
64 | }
65 |
66 | #endif /* TEST */
```

TEXT STATISTICS

```
  1379 characters
    66 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
    23 constants [numeric]
```

SYMBOL TABLE

TEST		47									
a	15	23	32	34							
argc		51	55								
argv		51	57								
atoi		57									
b	15	25	27	30	32						
c	15	26	36								
div		24	27	30	32	34	36	39	41	43	
div_t		16									
dy		53	60	61							
e	15	29	39								
easter		13	60								
easter_day			13	44							
easter_month			13	42	43						
f	16	24	25	26	27	28	29	30	31	32	33	34
	35	36	37	38	39	40	41	42	43	44		
g	15	31	32								
h	9	15	33	34	39	41	43	49			
i	15	37	39								
k	15	38	39								
main		51									
mo		53	60	61							
printf		61									
quot		25	28	31	35	37	42				
rem		26	29	33	38	40	44				
stdio		49									
stdlib		9									
u	15	35	39	41	43						
x	15	40	41	43							
year		13	23	24							
yr		53	57	58	59	60	61				
z	15	28	32								

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* editgets.c - line input w/editing                                    */
  4 | /* this code is released to the public domain                          */
  5 | /* written by Jon Burchmore                                            */
  6 | /* modifications & enhancements by Bob Stout                           */
  7 |
  8 | /* This is as close to ANSI compliant C that I could come, but it was made  */
  9 | /* on an IBM compatible computer, so I designed it for that platform.       */
 10 |
 11 | /* EXT_KEYS.C in SNIPPETS provides the ext_getch() function, which returns  */
 12 | /* a single extended character from the KEYBOARD, not stdin, and waits if   */
 13 | /* it must. It is be possible to re-write this function for a computer      */
 14 | /* besides an IBM PC. To do so, just rewrite EXT_KEYS.C and using the same  */
 15 | /* key definitions in EXT_KEYS.H                                            */
 16 |
 17 | /* Similarly, CURSOR.C in SNIPPETS is used to make the cursor larger when   */
 18 | /* in insert mode and smaller when in overwrite mode. To port to a non-PC   */
 19 | /* system, simply rewrite CURSOR.C using the same prototype and calling     */
 20 | /* parameters as specified in CURSOR.H.                                     */
 21 |
 22 | #include <conio.h>
 23 | #include <stdio.h>
 24 | #include <stdlib.h>
 25 | #include <string.h>
 26 | #include <ctype.h>
 27 | #include <time.h>
 28 | #include "sniptype.h"
 29 | #include "ext_keys.h"
 30 | #include "cursor.h"
 31 | #include "editgets.h"
 32 | #include "minmax.h"
 33 |
 34 | #define CTRL_CHARS_ALLOWED 0  /* Non-zero allows entry of Ctrl-characters   */
 35 |
 36 | #define BAD_KEY() {fputc('\a', stderr); break;}
 37 | #define putch(x)   fputc((x), stderr)
 38 |
 39 | int TabSize = 8;
 40 |
 41 | /*
 42 | **  Password mode:
 43 | **
 44 | **  0 - Normal operation
 45 | **  1 - Conventional password mode - '*' is echoed for all characters,
 46 | **      only ESC, ENTER, BACKSPC, and CTLHOME are active.
 47 | **  2 - Secure password mode - same as above except random characters are
 48 | **      displayed rather than '*'s to distract snoops from watching the keys.
 49 | */
 50 |
 51 | int password_mode = 0;
 52 |
 53 | static const char PWchars[] = /* Frequency approximates English usage   */
 54 |       "aaabbccddeeeeffgghhiiijkllmmmnnnoooppqrrrsssstttuuvwwxyz"
 55 |       "1234567890-|,@._"
 56 |       "aaabbccddeeeeffgghhiiijkllmmmnnnoooppqrrrsssstttuuvwwxyz";
 57 |
 58 | static int pickPWchar(void);
 59 |
 60 | /*
 61 | **  Arguments: 1) Buffer to receive string
 62 | **             2) Size of buffer
 63 | **             3) Default string
 64 | */
 65 |
 66 | int editgets(char *s, int maxlen, char *string)
 67 | {
 68 |       char temp[500];         /* Working buffer                        */
 69 |       int   insert,           /* Non-zero if in insert mode            */
 70 |             done = 0,        /* Non-Zero if done                      */
 71 |             pos,             /* Our position within the string        */
 72 |             len,             /* The current length of the string      */
 73 |             i, j, k,         /* Temporary variables                   */
 74 |             c,               /* Keyboard input                        */
 75 |             skip;            /* Spaces to skip when tabbing           */
 76 |
 77 |       if (NULL == string)
 78 |             string = "";
 79 |
 80 |       if (0 != (pos = len = strlen(string)))
 81 |             strncpy(temp, string, min(len, maxlen));
 82 |
 83 |       for (i = 0; i < maxlen; ++i)
 84 |       {
 85 |             if (NUL == *string)
 86 |                   putch('_');
 87 |             else  putch(*string++);
 88 |       }
 89 |       for (i = 0; i < (maxlen - len); ++i)
 90 |             putch('\b');
 91 |       fflush(stderr);
 92 |
 93 |       /* Set insert mode, save cursor, and use a big cursor */
 94 |
 95 |       insert = 1;
 96 |       cursor('S');
 97 |       cursor('B');
 98 |
 99 |       while (!done)
100 |       {
```

```
101 |             c = ext_getch();
102 |             switch (c)
103 |             {
104 |             case Key_ESC :
105 |                   if (len == 0)
106 |                         break;
107 |                   if (pos != len)
108 |                   {
109 |                         for (i = pos; i < len; i++)
110 |                               putch('_');
111 |                   }
112 |                   for (i = len; i > 0; i--)
113 |                   {
114 |                         putch('\b');
115 |                         putch('_');
116 |                         putch('\b');
117 |                   }
118 |                   pos = len = 0;
119 |                   break;
120 |
121 |             case Key_LTARROW :
122 |             case Key_PLTARROW :
123 |                   if (password_mode)
124 |                         BAD_KEY();
125 |                   if (pos == 0)
126 |                         break;
127 |                   pos--;
128 |                   putch('\b');
129 |                   break;
130 |
131 |             case Key_RTARROW :
132 |             case Key_PRTARROW :
133 |                   if (password_mode)
134 |                         BAD_KEY();
135 |                   if (pos == len)
136 |                         break;
137 |                   if (pos != maxlen)
138 |                   {
139 |                         putch(temp[pos]);
140 |                         pos++;
141 |                   }
142 |                   break;
143 |
144 |             case Key_HOME :
145 |             case Key_PHOME :
146 |                   if (password_mode)
147 |                         BAD_KEY();
148 |                   while (pos-- > 0)
149 |                         putch('\b');
150 |                   pos = 0;
151 |                   break;
152 |
153 |             case Key_END :
154 |             case Key_PEND :
155 |                   if (password_mode)
156 |                         BAD_KEY();
157 |                   while (pos < len)
158 |                         putch(temp[pos++]);
159 |                   break;
160 |
161 |             case Key_INS :
162 |             case Key_PINS :
163 |                   if (password_mode)
164 |                         BAD_KEY();
165 |                   insert = (!(insert));
166 |                   cursor(insert ? 'B' : 'N');
167 |                   break;
168 |
169 |             case Key_DEL :
170 |             case Key_PDEL :
171 |                   if (password_mode)
172 |                         BAD_KEY();
173 |                   if (pos == len)
174 |                         break;
175 |                   for (i = pos; i < len; i++)
176 |                         temp[i] = temp[i + 1];
177 |                   len--;
178 |                   for (i = pos; i < len; i++)
179 |                         putch(temp[i]);
180 |                   putch('_');
181 |                   for (i = len + 1; i > pos; i--)
182 |                         putch('\b');
183 |                   break;
184 |
185 |             case Key_BACKSPC :
186 |                   if (pos == 0)
187 |                         break;
188 |                   if (pos != len)
189 |                   {
190 |                         for (i = pos - 1; i < len; i++)
191 |                               temp[i] = temp[i + 1];
192 |                         pos--;
193 |                         len--;
194 |                         putch('\b');
195 |                         for (i = pos; i < len; i++)
196 |                               putch(temp[i]);
197 |                         putch('_');
198 |                         for (i = len; i >= pos; i--)
199 |                               putch('\b');
200 |                   }
```

```
201 |                   else
202 |                   {
203 |                         putch('\b');
204 |                         putch('_');
205 |                         putch('\b');
206 |                         pos = --len;
207 |                   }
208 |                   break;
209 |
210 |             case Key_ENTER :
211 |             case Key_PADENTER :
212 |             case Key_NL :
213 |                   done = 1;
214 |                   break;
215 |
216 |             case Key_CEND :
217 |             case Key_CPEND :
218 |                   if (password_mode)
219 |                         BAD_KEY();
220 |                   for (i = pos; i < len; ++i)
221 |                         putch('_');
222 |                   for (i = pos; i < len; ++i)
223 |                         putch('\b');
224 |                   len = pos;
225 |                   break;
226 |
227 |             case Key_CHOME :
228 |             case Key_CPHOME :
229 |                   if (pos == 0)
230 |                         break;
231 |                   if (pos != len)
232 |                   {
233 |                         while (0 != pos)
234 |                         {
235 |                               for (i = pos - 1; i < len; i++)
236 |                                     temp[i] = temp[i + 1];
237 |                               pos--;
238 |                               len--;
239 |                               putch('\b');
240 |                               for (i = pos; i < len; i++)
241 |                                     putch(temp[i]);
242 |                               putch('_');
243 |                               for (i = len; i >= pos; i--)
244 |                                     putch('\b');
245 |                         }
246 |                   }
247 |                   else
248 |                   {
249 |                         while (0 != pos)
250 |                         {
251 |                               putch('\b');
252 |                               putch('_');
253 |                               putch('\b');
254 |                               pos = --len;
255 |                         }
256 |                   }
257 |                   break;
258 |
259 |             case Key_CRTARROW :
260 |             case Key_CPRTARROW :
261 |                   if (password_mode)
262 |                         BAD_KEY();
263 |                   do
264 |                   {
265 |                         if (pos == len)
266 |                               break;
267 |                         if (pos != maxlen)
268 |                         {
269 |                               putch(temp[pos]);
270 |                               pos++;
271 |                         }
272 |                   } while (isspace(temp[pos]));
273 |                   do
274 |                   {
275 |                         if (pos == len)
276 |                               break;
277 |                         if (pos != maxlen)
278 |                         {
279 |                               putch(temp[pos]);
280 |                               pos++;
281 |                         }
282 |                   } while (!isspace(temp[pos]));
283 |                   break;
284 |
285 |             case Key_CLTARROW :
286 |             case Key_CPLTARROW :
287 |                   if (password_mode)
288 |                         BAD_KEY();
289 |                   do
290 |                   {
291 |                         if (pos == 0)
292 |                               break;
293 |                         pos--;
294 |                         putch('\b');
295 |                   } while (isspace(temp[pos]));
296 |                   do
297 |                   {
298 |                         if (pos == 0)
299 |                               break;
300 |                         pos--;
```

```
301 |                       putch('\b');
302 |                 } while (!isspace(temp[pos]));
303 |                 break;
304 |
305 |           case Key_TAB :
306 |                 if (password_mode)
307 |                       BAD_KEY();
308 |                 if (pos == maxlen)
309 |                       break;
310 |
311 |                 skip = TabSize - ((pos + TabSize) % TabSize);
312 |
313 |                 if (insert)
314 |                 {
315 |                       if ((len + skip) > maxlen)
316 |                             skip = maxlen - len;
317 |
318 |                       for (i = pos, j = len + skip - 1, k = len - 1;
319 |                             i < len ; ++i, --j, --k)
320 |                       {
321 |                             temp[j] = temp[k];
322 |                       }
323 |                 }
324 |                 else  /* overwrite */
325 |                 {
326 |                       if ((pos + skip) > maxlen)
327 |                             skip = maxlen - pos;
328 |                 }
329 |                 for (i = 0; i < skip; ++i)
330 |                 {
331 |                       temp[pos++] = ' ';
332 |                       putch(' ');
333 |                 }
334 |                 len = insert ? len + skip : max(len, pos);
335 |                 for (i = pos; i < len; ++i)
336 |                       putch(temp[i]);
337 |                 for (i = len; i > pos; --i)
338 |                       putch('\b');
339 |                 break;
340 |
341 |           default :
342 |                 if (pos == maxlen)
343 |                       break;
344 |
345 |                 if (c & 0xff00          /* Illegal extended character */
346 |
347 | #if !CTRL_CHARS_ALLOWED
348 |
349 |                 || iscntrl(c)
350 | #endif
351 |                 )
352 |                 {
353 |                       BAD_KEY();
354 |                 }
355 |
356 |                 if ((!(insert)) || pos == len)
357 |                 {
358 |                       temp[pos++] = (char)c;
359 |                       if (pos > len) len++;
360 |                       if (password_mode)
361 |                       {
362 |                             if (2 == password_mode)
363 |                                   putch(pickPWchar());
364 |                             else  putch('*');
365 |                       }
366 |                       else  putch(c);
367 |                 }
368 |                 else
369 |                 {
370 |                       if (len == maxlen)
371 |                             break;
372 |                       for (i = len++; i >= pos; i--)
373 |                             temp[i + 1] = temp[i];
374 |                       temp[pos++] = (char)c;
375 |                       if (password_mode)
376 |                       {
377 |                             if (2 == password_mode)
378 |                                   putch(pickPWchar());
379 |                             else  putch('*');
380 |                       }
381 |                       else  putch(c);
382 |                       for (i = pos; i < len; i++)
383 |                             putch(temp[i]);
384 |                       for (i = len; i > pos; i--)
385 |                             putch('\b');
386 |                 }
387 |           }
388 |       }
389 |       temp[len] = '\0';
390 |       strcpy(s, temp);
391 |       cursor('Z');            /* Restore cursor size  */
392 |       return len;
393 | }
394 |
395 | static int pickPWchar(void)
396 | {
397 |       static int inited = 0;
398 |
399 |       if (!inited)
400 |       {
```

```
401 |             srand(((unsigned int)time(NULL)) | 1);
402 |             inited = 1;
403 |       }
404 |
405 |       return PWchars[rand() % (sizeof(PWchars) - 1)];
406 | }
407 |
408 | #ifdef TEST
409 |
410 | main()
411 | {
412 |       char mystring[60];
413 |
414 |       memset(mystring, 0, 60);
415 |       puts("                    "
416 |             "0        1        2         3         4         5         6");
417 |       puts("                   "
418 |             "12345678901234567890123456789012345678901234567890123456789012345678901234567890");
419 |       fputs("Enter any string: ", stderr);
420 |       editgets(mystring, 60, "This is a test");
421 |       puts("");
422 |       printf("editgets() returned:\n\"%s\"\n", mystring);
423 |
424 |       password_mode = 1;
425 |       memset(mystring, 0, 60);
426 |       fputs("Enter any password: ", stderr);
427 |       editgets(mystring, 50, NULL);
428 |       puts("");
429 |       printf("editgets() returned:\n\"%s\"\n", mystring);
430 |
431 |       password_mode = 2;
432 |       memset(mystring, 0, 60);
433 |       fputs("Enter any password: ", stderr);
434 |       editgets(mystring, 50, NULL);
435 |       puts("");
436 |       printf("editgets() returned:\n\"%s\"\n", mystring);
437 |
438 |       return 0;
439 | }
440 |
441 | #endif /* TEST */
```

TEXT STATISTICS

14592 characters
441 lines

LEXICAL STATISTICS

33 comments [std-C]
0 comments [C++]
18 preprocessor instructions
39 constants [character]
23 constants [string]
52 constants [numeric]

SYMBOL TABLE

```
BAD_KEY          36     124     134     147     156     164     172     219     262     288     307
        353
CTRL_CHARS_ALLOWED              34     347
Key_BACKSPC             185
Key_CEND        216
Key_CHOME       227
Key_CLTARROW            285
Key_CPEND       217
Key_CPHOME              228
Key_CPLTARROW           286
Key_CPRTARROW           260
Key_CRTARROW            259
Key_DEL         169
Key_END         153
Key_ENTER       210
Key_ESC         104
Key_HOME        144
Key_INS         161
Key_LTARROW             121
Key_NL          212
Key_PADENTER            211
Key_PDEL        170
Key_PEND        154
Key_PHOME       145
Key_PINS        162
Key_PLTARROW            122
Key_PRTARROW            132
Key_RTARROW             131
Key_TAB         305
NUL              85
NULL             77     401     427     434
PWchars          53     405+
TEST            408
TabSize          39     311+
c        74     101     102     345     349     358     366     374     381
conio            22
ctype            26
cursor           96      97     166     391
done             70      99     213
editgets         66     420     427     434
ext_getch       101
fflush           91
fputc            36      37
fputs           419     426     433
h        22      23      24      25      26      27
i        73      83+     89+    109+    112+    175+    176+    178+    179     181+    190+    191+
        195+    196     198+    220+    222+    235+    236+    240+    241     243+    318     319+
        329+    335+    336     337+    372+    373+    382+    383     384+
inited          397     399     402
insert           69      95     165+    166     313     334     356
iscntrl         349
isspace         272     282     295     302
j        73     318     319     321
k        73     318     319     321
len              72      80      81      89     105     107     109     112     118     135     157
        173     175     177     178     181     188     190     193     195     198     206     220
        222     224     231     235     238     240     243     254     265     275     315     316
        318+    319     334+    335     337     356     359+    370     372     382     384     389
        392
main            410
max             334
maxlen           66      81      83      89     137     267     277     308     315     316     326
        327     342     370
memset          414     425     432
min              81
mystring        412     414     420     422     425     427     429     432     434     436
password_mode            51     123     133     146     155     163     171     218     261     287
        306     360     362     375     377     424     431
pickPWchar               58     363     378     395
pos              71      80     107     109     118     125     127     135     137     139     140
        148     150     157     158     173     175     178     181     186     188     190     192
        195     198     206     220     222     224     229     231     233     235     237     240
        243     249     254     265     267     269     270     272     275     277     279     280
        282     291     293     295     298     300     302     308     311     318     326     327
        331     334     335     337     342     356     358     359     372     374     382     384
printf          422     429     436
putch            37      86      87      90     110     114     115     116     128     139     149
        158     179     180     182     194     196     197     199     203     204     205     221
        223     239     241     242     244     251     252     253     269     279     294     301
        332     336     338     363     364     366     378     379     381     383     385
puts            415     417     421     428     435
rand            405
s        66     390
skip             75     311     315     316     318     326     327     329     334
```

```
srand           401
stderr           36      37      91      419     426     433
stdio            23
stdlib           24
strcpy          390
string           25      66      77      78      80      81      85      87
strlen           80
strncpy          81
temp             68      81      139     158     176+    179     191+    196     236+    241     269
       272      279     282     295     302     321+    331     336     358     373+    374     383
       389      390
time             27      401
x       37+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** jgets() - line input w/editing
  5 | */
  6 |
  7 | #ifndef EDITGETS__H
  8 | #define EDITGETS__H
  9 |
 10 | #include "minmax.h"
 11 |
 12 | extern int TabSize;
 13 |
 14 | /*
 15 | **  Password mode:
 16 | **
 17 | **  0 - Normal operation
 18 | **  1 - Conventional password mode - '*' is echoed for all characters,
 19 | **      only ESC, ENTER, BACKSPC, and CTLHOME are active.
 20 | **  2 - Secure password mode - same as above except random characters are
 21 | **      displayed rather than '*'s to distract snoops from watching the keys.
 22 | */
 23 |
 24 | extern int password_mode;
 25 |
 26 | /*
 27 | **  Line editing formatted text entry function.
 28 | */
 29 |
 30 | int editgets(char *s, int maxlen, char *string);
 31 |
 32 |
 33 | /*
 34 | **  Get a string of unknown length
 35 | */
 36 |
 37 | char *getstring(void);
 38 |
 39 |
 40 | #endif /* EDITGETS__H */
```

TEXT STATISTICS

```
   766 characters
    40 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    EDITGETS__H              7       8
    TabSize         12
    editgets        30
    getstring       37
    maxlen          30
    password_mode           24
    s       30
    string          30
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** EMS.C
**
** Expanded Memory Functions
**
** Released to the public domain by Cliff Rhodes with no guarantees
** of any kind.
*/

#include <stdio.h>
#include <dos.h>

#include "ems.h"

/* EMS driver values */
#define EMS_ID      "EMMXXXX0" /* EMS identifier string */
#define EMS_INT     0x67       /* EMS interrupt number */
#define EMS_VERSION 0x32       /* Version 3.2 of EMS */

/* EMS service codes */
#define EMSservice1  0x40    /* Get EMS status */
#define EMSservice2  0x41    /* Get segment address of page 0 */
#define EMSservice3  0x42    /* Get total number of expanded pages */
#define EMSservice4  0x43    /* Get handle and assign pages to it */
#define EMSservice5  0x44    /* Map a page into one of the page frames */
#define EMSservice6  0x45    /* Close EMS handle */
#define EMSservice7  0x46    /* Get the EMS version number */


/*
** Determines if EMS present. If so returns base segment of EMS.
** Returns 0 if EMS not available. The base segment is necessary
** for mapping EMS memory pages into the user address space (see
** EMSmap() below).
*/

unsigned int EMSbaseaddress(void)
{
      static const char tag[] = EMS_ID;

      char far      *p;
      int           i;
      union REGS    regs;
      struct SREGS  segs;

      regs.h.ah = 0x35;
      regs.h.al = EMS_INT;
      int86x(0x21, &regs, &regs, &segs);

      p = (char far *) (MK_FP(segs.es, 10)); /* EMS_ID must be at offset 10 */

      for(i = 0; i < 8; i++)
      {
            if(tag[i] != p[i])
                  return 0;                 /* If EMS_ID not found, return */
      }

      regs.h.ah = EMSservice2;              /* Get page frame segment */
      int86(EMS_INT, &regs, &regs);

      return regs.h.ah ? 0 : (unsigned int) regs.x.bx;
}

/*
** Returns current EMS version, -1 if not found or obsolete
*/

int EMSversion(void)
{
      union REGS regs;

      regs.h.ah = EMSservice7;
      int86(EMS_INT, &regs, &regs);

      if(!regs.h.ah && regs.h.al >= EMS_VERSION)  /* Make sure at least 3.2 */
            return regs.x.ax;

      return -1;
}

/*
** Returns 0 if EMS system OK, -1 if not
*/

int  EMSstatus(void)
{
      union REGS regs;

      regs.h.ah = EMSservice1;
      int86(EMS_INT, &regs, &regs);

      return regs.h.ah ? -1 : 0;
}

/*
** Returns number of free EMS pages (each page is 16k), -1 if error
*/

```

```
101 | int EMSpages(void)
102 | {
103 |       union REGS  regs;
104 |
105 |       regs.h.ah = EMSservice3;
106 |       int86(EMS_INT, &regs, &regs);
107 |
108 |       return regs.h.ah ? -1 : (int) regs.x.bx;
109 | }
110 |
111 | /*
112 | ** Returns handle to block of size pages or -1 if error.
113 | ** NOTE: always free any handles when you are done!.
114 | */
115 |
116 | int EMSalloc(int pages)
117 | {
118 |       union REGS regs;
119 |
120 |       regs.h.ah = EMSservice4;
121 |       regs.x.bx = pages;
122 |       int86(EMS_INT, &regs, &regs);
123 |
124 |       return regs.h.ah ? -1 : (int) regs.x.dx;
125 | }
126 |
127 | /*
128 | ** Frees handle block, returns 0 if successful, -1 if error
129 | */
130 |
131 | int EMSfree(int handle)
132 | {
133 |       union REGS regs;
134 |
135 |       regs.h.ah = EMSservice6;
136 |       regs.x.dx = handle;
137 |       int86(EMS_INT, &regs, &regs);
138 |
139 |       return regs.h.ah ? -1 : 0;
140 | }
141 |
142 | /*
143 | ** Maps page of handle into bank. Returns 0 if successful, -1 if error.
144 | ** Each handle controls 1 or more 16k pages of EMS memory.
145 | ** There are four banks 0-3. bank 0 starts at the segment returned by
146 | ** EMSbaseaddress(), bank 1 starts at that segment with offset 16k, etc.
147 | */
148 |
149 | int  EMSmap(int bank, int handle, int page)
150 | {
151 |       union REGS regs;
152 |
153 |       regs.h.ah = EMSservice5;
154 |       regs.h.al = bank;
155 |       regs.x.bx = page;
156 |       regs.x.dx = handle;
157 |       int86(EMS_INT, &regs, &regs);
158 |
159 |       return regs.h.ah ? -1 : 0;
160 | }
```

TEXT STATISTICS

```
  3797 characters
   160 lines
```

LEXICAL STATISTICS

```
    25 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     2 constants [string]
    25 constants [numeric]
```

SYMBOL TABLE

```
    EMS_ID          18      41
    EMS_INT         19      49      61      75      92     106     122     137     157
    EMS_VERSION             20      77
    EMSalloc       116
    EMSbaseaddress          39
    EMSfree        131
    EMSmap         149
    EMSpages       101
    EMSservice1             23      91
    EMSservice2             24      60
    EMSservice3             25     105
    EMSservice4             26     120
    EMSservice5             27     153
    EMSservice6             28     135
    EMSservice7             29      74
    EMSstatus       87
    EMSversion              70
    MK_FP           52
    REGS            45      72      89     103     118     133     151
    SREGS           46
    ah              48      60      63      74      77      91      94     105     108     120     124
           135     139     153     159
    al              49      77     154
    ax              78
    bank           149     154
    bx              63     108     121     155
    dos             13
    dx             124     136     156
    es              52
    far             43      52
    h       12      13      48      49      60      63      74      77+     91      94     105     108
           120     124     135     139     153     154     159
    handle         131     136     149     156
    i       44      54+     56+
    int86           61      75      92     106     122     137     157
    int86x          50
    p       43      52      56
    page           149     155
    pages          116     121
    regs            45      48      49      50+     60      61+     63+     72      74      75+     77+
            78      89      91      92+     94     103     105     106+    108+    118     120     121
           122+    124+    133     135     136     137+    139     151     153     154     155     156
           157+    159
    segs            46      50      52
    stdio           12
    tag             41      56
    x       63      78     108     121     124     136     155     156
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** EMS.H
  5 | **
  6 | ** Header file Expanded Memory Routines
  7 | */
  8 |
  9 | #ifndef EMS_H_
 10 | #define EMS_H_
 11 |
 12 | #define EMS_PAGE_SIZE   16384   /* Each page is this size */
 13 |
 14 | unsigned int EMSbaseaddress(void);
 15 | int  EMSversion(void);
 16 | int  EMSstatus(void);
 17 | int  EMSpages(void);
 18 | int  EMSalloc(int pages);
 19 | int  EMSfree(int handle);
 20 | int  EMSmap(int bank, int handle, int page);
 21 |
 22 | #endif
```

TEXT STATISTICS

```
   422 characters
    22 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    EMS_H_           9      10
    EMS_PAGE_SIZE           12
    EMSalloc        18
    EMSbaseaddress          14
    EMSfree         19
    EMSmap          20
    EMSpages        17
    EMSstatus       16
    EMSversion              15
    bank            20
    handle          19      20
    page            20
    pages           18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** EMSTEST.C
**
** Test EMS functions found in ems.c
**
** Released to the public domain by Cliff Rhodes with no guarantees
** of any kind.
*/

#include <stdio.h>
#include <dos.h>

#include "ems.h"

int main(void)
{
      const char str[] = "This is just a test string!";
      int i, ver, handle1, handle2;
      unsigned int baseaddress;
      char far *cp;

      /* Get the segment for mapping expanded memory into address space */

      if((baseaddress = EMSbaseaddress()) == 0)
      {
            printf("Expanded memory manager not found, "
                  "terminating program.\n");
            return 1;
      }

      /* Get the version number an the number of pages available */

      ver = EMSversion();
      printf("EMM Version %d.%d loaded\n", ver >> 4, ver & 0x0f);
      printf("There are %d pages available\n", EMSpages());

      /* Allocate a few pages */

      handle1 = EMSalloc(3);   /* Allocate 3 pages */
      if(handle1 < 0)
      {
            printf("EMM allocation failed on handle 1\n");
            return 1;
      }
      handle2 = EMSalloc(1);   /* Allocate 1 page */
      if(handle2 < 0)
      {
            EMSfree(handle1);
            printf("EMM allocation failed on handle 2\n");
            return 1;
      }
      printf("There are %d pages available after allocating 4\n", EMSpages());

      /* Map the allocated pages into the address space */

      if(EMSmap(0, handle1, 1) < 0)            /* Mapped page 1 into bank 0 */
            printf("Error mapping handle 1\n");
      if(EMSmap(3, handle2, 0) < 0)            /* Mapped page 0 into bank 3 */
            printf("Error mapping handle 2\n");
      else
      {
            /* Write test string into beginning of handle2 */

            cp = (char far *) (MK_FP(baseaddress, 3 * EMS_PAGE_SIZE));
            for(i = 0; str[i]; i++)
                  cp[i] = str[i];
            cp[i] = '\0';

            /* Read back test string and look for a match */

            cp = (char far *) (MK_FP(baseaddress, 3 * EMS_PAGE_SIZE));
            for(i = 0; str[i]; i++)
            {
                  if(cp[i] != str[i])
                  {
                        printf("Data mismatch at character %d\n", i);
                        break;
                  }
            }
      }
      EMSfree(handle1);
      EMSfree(handle2);

      return 0;
}
```

TEXT STATISTICS

```
  2485 characters
    87 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     1 constants [character]
    12 constants [string]
    21 constants [numeric]
```

SYMBOL TABLE

```
    EMS_PAGE_SIZE           66      73
    EMSalloc        41      47
    EMSbaseaddress          26
    EMSfree         50      83      84
    EMSmap          58      60
    EMSpages        37      54
    EMSversion              35
    MK_FP           66      73
    baseaddress             21      26      66      73
    cp              22      66      68      69      73      76
    dos             13
    far             22      66      73
    h       12      13
    handle1         20      41      42      50      58      83
    handle2         20      47      48      60      84
    i       20      67+     68+     69      74+     76+     78
    main            17
    printf          28      36      37      44      51      54      59      61      78
    stdio           12
    str             19      67      68      74      76
    ver             20      35      36+
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /* ENG.C - Format floating point in engineering notation          */
 4 | /* Released to public domain by author, David Harmon, Jan. 1994   */
 5 |
 6 | #include <stdio.h>
 7 | #include "snipmath.h"
 8 |
 9 | char *eng(double value, int places)
10 | {
11 |       const char * const prefixes[] = {
12 |             "a", "f", "p", "n", "æ", "m", "", "k", "M", "G", "T"
13 |             };
14 |       int p = 6;
15 |       static char result[30];
16 |       char *res = result;
17 |
18 |       if (value < 0.)
19 |       {
20 |             *res++ = '-';
21 |             value = -value;
22 |       }
23 |       while (value != 0 && value < 1. && p > 0)
24 |       {
25 |             value *= 1000.;
26 |             p--;
27 |       }
28 |       while (value != 0 && value > 1000. && p < 10 )
29 |       {
30 |             value /= 1000.;
31 |             p++;
32 |       }
33 |       if (value > 100.)
34 |             places--;
35 |       if (value > 10.)
36 |             places--;
37 |       sprintf(res, "%.*f %s", places-1, value, prefixes[p]);
38 |       return result;
39 | }
40 |
41 | #ifdef TEST
42 |
43 | #include <stdio.h>
44 |
45 | main()
46 | {
47 |       double w;
48 |
49 |       for (w = 1e-19; w < 1e16; w *= 42)
50 |             printf(" %g W = %sW\n", w, eng(w, 3));
51 |       return 0;
52 | }
53 | #endif /* TEST */
```

TEXT STATISTICS

```
  1186 characters
    53 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
    14 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    TEST            41
    eng              9      50
    h        6      43
    main            45
    p       14      23      26      28      31      37
    places           9      34      36      37
    prefixes        11      37
    printf          50
    res             16      20      37
    result          15      16      38
    sprintf         37
    stdio            6      43
    value            9      18      21+     23+     25      28+     30      33      35      37
    w       47      49+     50+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** ERRFIX.C - redirect stderr to some other file under MS-DOS
**
** by Bob Jarvis
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include "extkword.h"

#ifdef __SC__
 #define SVP_CAST(x) (const char * const *)(x)
#else
 #define SVP_CAST(x) (x)
#endif

int usage(void)
{
      puts("ERRFIX [filename] [prog] { {parm1} {parm2} ... {parmN} }");
      puts("   Redirects stderr to another file, then invokes a program");
      puts("   which will inherit the new definition of stderr.\n");
      puts("Parameters:");
      puts("   filename (required) - the name of the file stderr should");
      puts("      be redirected to.  Output written to stderr will");
      puts("      be routed to this file instead of the console.");
      puts("   prog (required) - name of the program to be run.");
      puts("   parm1...parmN (optional) - command-line parameters needed");
      puts("      to run the program specified by the 'prog' argument.");
      return 1;
}

int main(int argc, char *argv[])
{
      char **args = argv;

      if (3 > argc)
            return usage();

      if (NULL != argv[argc]) /* may be a problem under some compilers */
      {
            args = malloc((argc+1) * sizeof(char *));
            if (NULL == args)
            {
                  printf("Unable to allocate storage");
                  return 2;
            }

            memcpy(args, argv, argc * sizeof(char *));

            args[argc] = NULL;
      }

      freopen(args[1], "w", stderr);

      spawnvp(0, args[2], SVP_CAST(&args[2]));

      return 0;
}
```

TEXT STATISTICS

```
  1696 characters
    62 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
    13 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    NULL            43      46      54
    SVP_CAST        16      18      59
    __SC__          15
    argc            36      40      43      45      52      54
    args            38      45      46      52      54      57      59+
    argv            36      38      43      52
    freopen         57
    h        9      10      11      12
    main            36
    malloc          45
    memcpy          52
    printf          48
    process         12
    puts            23      24      25      26      27      28      29      30      31      32
    spawnvp         59
    stderr          57
    stdio            9
    stdlib          10
    string          11
    usage           21      41
    x       16+     18+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Header file for SNIPPETS error handlers
  5 | */
  6 |
  7 | #ifndef ERRORS__H
  8 | #define ERRORS__H
  9 |
 10 | #include <stdio.h>
 11 |
 12 | int  ferrorf(FILE *filehandle, const char *format, ...); /* Ferrorf.C   */
 13 | FILE * cant(char *fname, char *fmode);                   /* Ferrorf.C   */
 14 | void ErrExit(char *fmt, ...);                            /* Err_Exit.C  */
 15 |
 16 | #endif /* ERRORS__H */
```

TEXT STATISTICS

```
   415 characters
    16 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    ERRORS__H        7       8
    ErrExit         14
    FILE            12      13
    cant            13
    ferrorf         12
    filehandle              12
    fmode           13
    fmt             14
    fname           13
    format          12
    h       10
    stdio           10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  ERR_EXIT.C - A generic fatal error-handler by Dave Schaumann
  5 | */
  6 |
  7 | #include <stdarg.h>
  8 | #include <stdio.h>
  9 | #include <stdlib.h>
 10 | #include "errors.h"
 11 |
 12 | void ErrExit(char *fmt, ...)
 13 | {
 14 |       va_list ap;
 15 |
 16 |       va_start(ap, fmt) ;
 17 |       vfprintf(stderr, fmt, ap);
 18 |       fputc('\n', stderr);
 19 |       exit(EXIT_FAILURE);
 20 | }
 21 |
 22 | #ifdef TEST
 23 |
 24 | main()
 25 | {
 26 |       int x = 1, y = -1;
 27 |
 28 |       if (x != y)
 29 |             ErrExit("Found x = %d, y = %d; Expected them to be equal!", x, y);
 30 |       return EXIT_SUCCESS;
 31 | }
 32 |
 33 | #endif /* TEST */
```

TEXT STATISTICS

```
   585 characters
    33 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     1 constants [character]
     2 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            19
    EXIT_SUCCESS            30
    ErrExit         12      29
    TEST            22
    ap              14      16      17
    exit            19
    fmt             12      16      17
    fputc           18
    h        7       8       9
    main            24
    stdarg           7
    stderr          17      18
    stdio            8
    stdlib           9
    va_list         14
    va_start        16
    vfprintf        17
    x       26      28      29
    y       26      28      29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ETPHI.c
**
**  Compute Euler's Totient function, phi().
**  phi(n) = the number of positive integers less than n
**  which have no common factors with n
**
**  Written by M. Stapleton of Graphic Bits
**  Public Domain
**
**  Sep  7 1996
**
**  Uses ISQRT.C, also in SNIPPETS
*/

#include "snipmath.h"

#undef ULONG
typedef unsigned long ULONG;

ULONG intsqrt(ULONG n);

static void dofact(ULONG i);
ULONG etphi(ULONG n);

static ULONG num, ph;

void dofact(ULONG i)
{
      long p;

      for (p = 0; (num % i) == 0; p++)
            num /= i;

      if (p)
      {
            ph *= i - 1;
            while(--p)
                  ph *= i;
      }
}

ULONG etphi(ULONG n)
{
      ULONG i;
      struct int_sqrt mi;

      if (n<2)
            return 0;

      num = n;
      ph = 1;
      usqrt(n, &mi);

      dofact(2);
      dofact(3);
      for (i = 5; (i <= mi.sqrt) && (num >= i); i += 6)
      {
            dofact(i);
            if (num < i + 2)
                  break;
            dofact(i + 2);
      }

      if (num > 1)
            ph *= num - 1;

      return ph;
}

#ifdef TEST

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
      long n = 0;

      if (argc > 1)
            n = atol(argv[1]);

      printf("etphi(%ld) = %ld\n", n, etphi(n));

      return 0;
}

#endif
```

TEXT STATISTICS

```
  1458 characters
    90 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     2 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    TEST            73
    ULONG           20      21      23+     25      26+     28      30      45+     47
    argc            78      82
    argv            78      83
    atol            83
    dofact          25      30      57      58      61      64
    etphi           26      45      85
    h       75      76
    i       25      30      34      35      39      41      47      59+     61      62      64
    int_sqrt        48
    intsqrt         23
    main            78
    mi              48      55      59
    n       23      26      45      50      53      55      80      83      85+
    num             28      34      35      53      59      62      67      68
    p       32      34+     37      40
    ph              28      39      41      54      68      70
    printf          85
    sqrt            59
    stdio           76
    stdlib          75
    usqrt           55
```

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************************/
/*                                                                       */
/*  EVAL.C - A simple mathematical expression evaluator in C             */
/*                                                                       */
/*  operators supported: Operator               Precedence               */
/*                                                                       */
/*                          (                    Lowest                  */
/*                          )                    Highest                 */
/*                          +   (addition)       Low                     */
/*                          -   (subtraction)    Low                     */
/*                          *   (multiplication) Medium                  */
/*                          /   (division)       Medium                  */
/*                          \   (modulus)        High                    */
/*                          ^   (exponentiation) High                    */
/*                          sin(                 Lowest                  */
/*                          cos(                 Lowest                  */
/*                          atan(                Lowest                  */
/*                          abs(                 Lowest                  */
/*                          sqrt(                Lowest                  */
/*                          ln(                  Lowest                  */
/*                          exp(                 Lowest                  */
/*                                                                       */
/*  constants supported: pi                                              */
/*                                                                       */
/*  Original Copyright 1991-93 by Robert B. Stout as part of             */
/*  the MicroFirm Function Library (MFL)                                 */
/*                                                                       */
/*  The user is granted a free limited license to use this source file   */
/*  to create royalty-free programs, subject to the terms of the         */
/*  license restrictions specified in the LICENSE.MFL file.              */
/*                                                                       */
/*  Requires RMALLWS.C, also in SNIPPETS.                                */
/*                                                                       */
/*************************************************************************/

#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "sniptype.h"
#include "snip_str.h"                    /* For rmallws(), strupr()    */
#include "snipmath.h"
#include "numcnvrt.h"

/*
**  Other SNIPPETS functions
*/

char *rmallws(char *);
char *strupr(char *);


struct operator {
      char        token;
      char       *tag;
      size_t      taglen;
      int         precedence;
};

static struct operator verbs[] = {
      {'+',  "+",       1, 2 },
      {'-',  "-",       1, 3 },
      {'*',  "*",       1, 4 },
      {'/',  "/",       1, 5 },
      {'\\', "\\",      1, 5 },
      {'^',  "^",       1, 6 },
      {'(',  "(",       1, 0 },
      {')',  ")",       1, 99},
      {'S',  "SIN(",    4, 0 },
      {'C',  "COS(",    4, 0 },
      {'A',  "ABS(",    4, 0 },
      {'L',  "LN(",     3, 0 },
      {'E',  "EXP(",    4, 0 },
      {'t',  "ATAN(",   5, 0 },
      {'s',  "SQRT(",   5, 0 },
      {NUL,  NULL,      0, 0 }
};

static char   op_stack[256];                  /* Operator stack       */
static double arg_stack[256];                 /* Argument stack       */
static char   token[256];                     /* Token buffer         */
static int    op_sptr,                        /* op_stack pointer     */
              arg_sptr,                       /* arg_stack pointer    */
              parens,                         /* Nesting level        */
              state;                          /* 0 = Awaiting expression
                                                 1 = Awaiting operator
                                              */
const double Pi = 3.14159265358979323846;

static int              do_op(void);
static int              do_paren(void);
static void             push_op(char);
static void             push_arg(double);
static int              pop_arg(double *);
static int              pop_op(int *);
static char            *get_exp(char *);
static struct operator *get_op(char *);
static int              getprec(char);
```

```
101 | static int              getTOSprec(void);
102 |
103 | /*************************************************************************/
104 | /*                                                                       */
105 | /*  evaluate()                                                           */
106 | /*                                                                       */
107 | /*  Evaluates an ASCII mathematical expression.                          */
108 | /*                                                                       */
109 | /*  Arguments: 1 - String to evaluate                                    */
110 | /*             2 - Storage to receive double result                      */
111 | /*                                                                       */
112 | /*  Returns: Success_ if successful                                      */
113 | /*           Error_ if syntax error                                      */
114 | /*           R_ERROR if runtime error                                    */
115 | /*                                                                       */
116 | /*  Side effects: Removes all whitespace from the string and converts    */
117 | /*                it to U.C.                                             */
118 | /*                                                                       */
119 | /*************************************************************************/
120 |
121 | int evaluate(char *line, double *val)
122 | {
123 |       double arg;
124 |       char *ptr = line, *str, *endptr;
125 |       int ercode;
126 |       struct operator *op;
127 |
128 |       strupr(line);
129 |       rmallws(line);
130 |       state = op_sptr = arg_sptr = parens = 0;
131 |
132 |       while (*ptr)
133 |       {
134 |             switch (state)
135 |             {
136 |             case 0:
137 |                   if (NULL != (str = get_exp(ptr)))
138 |                   {
139 |                         if (NULL != (op = get_op(str)) &&
140 |                               strlen(str) == op->taglen)
141 |                         {
142 |                               push_op(op->token);
143 |                               ptr += op->taglen;
144 |                               break;
145 |                         }
146 |
147 |                         if (Success_ == strcmp(str, "-"))
148 |                         {
149 |                               push_op(*str);
150 |                               ++ptr;
151 |                               break;
152 |                         }
153 |
154 |                         if (Success_ == strcmp(str, "PI"))
155 |                               push_arg(Pi);
156 |
157 |                         else
158 |                         {
159 |                               if (0.0 == (arg = strtod(str, &endptr)) &&
160 |                                     NULL == strchr(str, '0'))
161 |                               {
162 |                                     return Error_;
163 |                               }
164 |                               push_arg(arg);
165 |                         }
166 |                         ptr += strlen(str);
167 |                   }
168 |                   else  return Error_;
169 |
170 |                   state = 1;
171 |                   break;
172 |
173 |             case 1:
174 |                   if (NULL != (op = get_op(ptr)))
175 |                   {
176 |                         if (')' == *ptr)
177 |                         {
178 |                               if (Success_ > (ercode = do_paren()))
179 |                                     return ercode;
180 |                         }
181 |                         else
182 |                         {
183 |                               while (op_sptr &&
184 |                                     op->precedence <= getTOSprec())
185 |                               {
186 |                                     do_op();
187 |                               }
188 |                               push_op(op->token);
189 |                               state = 0;
190 |                         }
191 |
192 |                         ptr += op->taglen;
193 |                   }
194 |                   else  return Error_;
195 |
196 |                   break;
197 |             }
198 |       }
199 |
200 |       while (1 < arg_sptr)
```

```
201 |       {
202 |             if (Success_ > (ercode = do_op()))
203 |                   return ercode;
204 |       }
205 |       if (!op_sptr)
206 |             return pop_arg(val);
207 |       else  return Error_;
208 | }
209 |
210 | /*
211 | **  Evaluate stacked arguments and operands
212 | */
213 |
214 | static int do_op(void)
215 | {
216 |       double arg1, arg2;
217 |       int op;
218 |
219 |       if (Error_ == pop_op(&op))
220 |             return Error_;
221 |
222 |       pop_arg(&arg1);
223 |       pop_arg(&arg2);
224 |
225 |       switch (op)
226 |       {
227 |       case '+':
228 |             push_arg(arg2 + arg1);
229 |             break;
230 |
231 |       case '-':
232 |             push_arg(arg2 - arg1);
233 |             break;
234 |
235 |       case '*':
236 |             push_arg(arg2 * arg1);
237 |             break;
238 |
239 |       case '/':
240 |             if (0.0 == arg1)
241 |                   return R_ERROR;
242 |             push_arg(arg2 / arg1);
243 |             break;
244 |
245 |       case '\\':
246 |             if (0.0 == arg1)
247 |                   return R_ERROR;
248 |             push_arg(fmod(arg2, arg1));
249 |             break;
250 |
251 |       case '^':
252 |             push_arg(pow(arg2, arg1));
253 |             break;
254 |
255 |       case 't':
256 |             ++arg_sptr;
257 |             push_arg(atan(arg1));
258 |             break;
259 |
260 |       case 'S':
261 |             ++arg_sptr;
262 |             push_arg(sin(arg1));
263 |             break;
264 |
265 |       case 's':
266 |             if (0.0 > arg2)
267 |                   return R_ERROR;
268 |             ++arg_sptr;
269 |             push_arg(sqrt(arg1));
270 |             break;
271 |
272 |       case 'C':
273 |             ++arg_sptr;
274 |             push_arg(cos(arg1));
275 |             break;
276 |
277 |       case 'A':
278 |             ++arg_sptr;
279 |             push_arg(fabs(arg1));
280 |             break;
281 |
282 |       case 'L':
283 |             if (0.0 < arg1)
284 |             {
285 |                   ++arg_sptr;
286 |                   push_arg(log(arg1));
287 |                   break;
288 |             }
289 |             else  return R_ERROR;
290 |
291 |       case 'E':
292 |             ++arg_sptr;
293 |             push_arg(exp(arg1));
294 |             break;
295 |
296 |       case '(':
297 |             arg_sptr += 2;
298 |             break;
299 |
300 |       default:
```

```
301 |             return Error_;
302 |       }
303 |       if (1 > arg_sptr)
304 |             return Error_;
305 |       else  return op;
306 | }
307 |
308 | /*
309 | **  Evaluate one level
310 | */
311 |
312 | static int do_paren(void)
313 | {
314 |       int op;
315 |
316 |       if (1 > parens--)
317 |             return Error_;
318 |       do
319 |       {
320 |             if (Success_ > (op = do_op()))
321 |                   break;
322 |       } while (getprec((char)op));
323 |       return op;
324 | }
325 |
326 | /*
327 | **  Stack operations
328 | */
329 |
330 | static void push_op(char op)
331 | {
332 |       if (!getprec(op))
333 |             ++parens;
334 |       op_stack[op_sptr++] = op;
335 | }
336 |
337 | static void push_arg(double arg)
338 | {
339 |       arg_stack[arg_sptr++] = arg;
340 | }
341 |
342 | static int pop_arg(double *arg)
343 | {
344 |       *arg = arg_stack[--arg_sptr];
345 |       if (0 > arg_sptr)
346 |             return Error_;
347 |       else  return Success_;
348 | }
349 |
350 | static int pop_op(int *op)
351 | {
352 |       if (!op_sptr)
353 |             return Error_;
354 |       *op = op_stack[--op_sptr];
355 |       return Success_;
356 | }
357 |
358 | /*
359 | **  Get an expression
360 | */
361 |
362 | static char * get_exp(char *str)
363 | {
364 |       char *ptr = str, *tptr = token;
365 |       struct operator *op;
366 |
367 |       if (Success_ == strncmp(str, "PI", 2))
368 |             return strcpy(token, "PI");
369 |
370 |
371 |       while (*ptr)
372 |       {
373 |             if (NULL != (op = get_op(ptr)))
374 |             {
375 |                   if ('-' == *ptr)
376 |                   {
377 |                         if (str != ptr && 'E' != ptr[-1])
378 |                               break;
379 |                         if (str == ptr && !isdigit(ptr[1]) && '.' != ptr[1])
380 |                         {
381 |                               push_arg(0.0);
382 |                               strcpy(token, op->tag);
383 |                               return token;
384 |                         }
385 |                   }
386 |
387 |                   else if (str == ptr)
388 |                   {
389 |                         strcpy(token, op->tag);
390 |                         return token;
391 |                   }
392 |
393 |                   else break;
394 |             }
395 |
396 |             *tptr++ = *ptr++;
397 |       }
398 |       *tptr = NUL;
399 |
400 |       return token;
```

```
}

/*
**  Get an operator
*/

static struct operator * get_op(char *str)
{
      struct operator *op;

      for (op = verbs; op->token; ++op)
      {
            if (Success_ == strncmp(str, op->tag, op->taglen))
                  return op;
      }
      return NULL;
}

/*
**  Get precedence of a token
*/

static int getprec(char token)
{
      struct operator *op;

      for (op = verbs; op->token; ++op)
      {
            if (token == op->token)
                  break;
      }
      if (op->token)
            return op->precedence;
      else  return 0;
}

/*
**  Get precedence of TOS token
*/

static int getTOSprec(void)
{
      if (!op_sptr)
            return 0;
      return getprec(op_stack[op_sptr - 1]);
}

#ifdef TEST

#include <stdio.h>

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif
#ifdef __TURBOC__
 #pragma argsused
#endif

main(int argc, char *argv[])
{
      int retval;
      double val;

      printf("evaluate(%s) ", argv[1]);
      printf("returned %d\n", retval = evaluate(argv[1], &val));
      if (0 == retval)
            printf("val = %f\n", val);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

13790 characters
471 lines

LEXICAL STATISTICS

69 comments [std-C]
0 comments [C++]
17 preprocessor instructions
34 constants [character]
26 constants [string]
63 constants [numeric]

SYMBOL TABLE

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************************/
/*                                                                       */
/*  EXISTSX.C                                                            */
/*                                                                       */
/*  POSIX-compliant functions to verify the existance of files in the    */
/*  PATH or in other environment variables.                              */
/*                                                                       */
/*  Original Copyright 1990-93 by Robert B. Stout as part of             */
/*  the MicroFirm Function Library (MFL)                                 */
/*                                                                       */
/*  The user is granted a free limited license to use this source file   */
/*  to create royalty-free programs, subject to the terms of the         */
/*  license restrictions specified in the LICENSE.MFL file.              */
/*                                                                       */
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif
#include "snipfile.h"
#include "snip_str.h"

/*
**  exists()
**
**  See if a file exists.
**
**  Parameters: 1 - File name
**
**  Returns: True_ if found, else False_.
**
*/

Boolean_T exists(char *name)
{
      return(Success_ == access(name, F_OK));
}

/*
**  dexists()
**
**  See if a file exists in an environment variable.
**
**  Parameters: 1 - File name
**              2 - Environment variable name
**
**  Returns: The pathname of the file which was found or NULL if the file
**           was not found.
**
**  Side effects: The returned path name is a static string. Subesequent
**                calls to any of these [d/p/g]exists() functions will
**                destroy the previous contents.
**
**  Notes: Uses stptok() and dos2unix(), also in SNIPPETS. May require
**         strupr() from SNIPPETS if it's not in your standard library.
*/

char *dexists(char *name, char *envar)
{
      char *path, *ptr;
      static char newname[FILENAME_MAX];
      int i, j;
      char chr;

      if (exists(name))
            return(name);

      if (NULL == (path = getenv(strupr(envar))))
            return(NULL);                       /* no environment var.  */

      dos2unix(path);

      do
      {
            path = stptok(path, newname, FILENAME_MAX, ";");
            if (LAST_CHAR(newname) != '/')
                  strcat(newname, "/");
            strcat(newname, name);
            if (exists(newname))
                  return(newname);
      } while (path && *path);

      return NULL;
}

/*
**  pexists()
**
**  See if a file exists in the PATH.
**
**  Parameters: 1 - File name
**
**  Returns: The pathname of the file which was found or NULL if the file
```

```
101 | **           was not found.
102 | **
103 | **  Side effects: The returned path name is a static string. Subesequent
104 | **                calls to any of these [d/p/g]exists() functions will
105 | **                destroy the previous contents.
106 | */
107 |
108 | char *pexists(char *name)
109 | {
110 |       return dexists(name, "PATH");
111 | }
112 |
113 | /*
114 | **  gexists()
115 | **
116 | **  See if a file exists in the PATH or in an environment variable.
117 | **
118 | **  Parameters: 1 - File name
119 | **              2 - Environment variable name
120 | **
121 | **  Returns: The pathname of the file which was found or NULL if the file
122 | **           was not found.
123 | **
124 | **  Side effects: The returned path name is a static string. Subesequent
125 | **                calls to any of these [d/p/g]exists() functions will
126 | **                destroy the previous contents.
127 | */
128 |
129 | char *gexists(char *name, char *envar)
130 | {
131 |       char *tmp;
132 |
133 |       if (exists(name))
134 |             return(name);
135 |       if (NULL != (tmp = dexists(name, envar)))
136 |             return(tmp);
137 |       return(pexists(name));
138 | }
139 |
140 | #ifdef TEST
141 |
142 | main(int argc, char *argv[])
143 | {
144 |       char *fname;
145 |
146 |       switch (argc)
147 |       {
148 |       case 2:
149 |             fname = pexists(argv[1]);
150 |             if (fname)
151 |                   printf("%s found in PATH\n", fname);
152 |             else  printf("%s not found in PATH\n", argv[1]);
153 |             break;
154 |       case 3:
155 |             fname = gexists(argv[1], argv[2]);
156 |             if (fname)
157 |                   printf("%s found in PATH or %s\n", fname, argv[2]);
158 |             else  printf("%s not found in PATH or %s\n",
159 |                         argv[1], argv[2]);
160 |             break;
161 |       default:
162 |             puts("\aUsage: EXISTSX fname [envar]");
163 |             return EXIT_FAILURE;
164 |       }
165 |
166 |       return EXIT_SUCCESS;
167 | }
168 |
169 | #endif /* TEST */
```

```
TEXT STATISTICS

  4894 characters
   169 lines

LEXICAL STATISTICS

    22 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     1 constants [character]
    11 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    Boolean_T       41
    EXIT_FAILURE           163
    EXIT_SUCCESS           166
    FILENAME_MAX            68      82
    F_OK            43
    LAST_CHAR       83
    MSDOS           22
    NULL            75      76      90     135
    Success_        43
    TEST           140
    __MSDOS__       22
    access          43
    argc           142     146
    argv           142     149     152     155+    157     159+
    chr             70
    defined         22+
    dexists         65     110     135
    dos2unix        78
    envar           65      75     129     135
    exists          41      72      86     133
    fname          144     149     150     151     155     156     157
    getenv          75
    gexists        129     155
    h       19      20      21      25
    i       69
    j       69
    main           142
    name            41      43      65      72      73      85     108     110     129     133     134
           135     137
    newname         68      82      83      84      85      86      87
    path            67      75      78      82+     88+
    pexists        108     137     149
    printf         151     152     157     158
    ptr             67
    puts           162
    stdio           19
    stdlib          20
    stptok          82
    strcat          84      85
    string          21
    strupr          75
    tmp            131     135     136
    unistd          25
```

```
/* +++Date last modified: 05-Jul-1997 */

/*========================================================================

    _MSC_VER        Microsoft C 6.0 and later
    _QC             Microsoft Quick C 2.51 and later
    __TURBOC__      Borland Turbo C, Turbo C++ and BC++
    __BORLANDC__    Borland C++
    __ZTC__         Zortech C and C++
    __SC__          Symantec C++
    __WATCOMC__     WATCOM C
    __POWERC        Mix Power C
    __GNUC__        Gnu C

    Revised:

    25-Sep-95  Bob Stout      Original from PC-PORT.H
    30-Mar-96  Ed Blackman  OS/2 mods for OS/2 ver 2.0 and up
    30-May-96  Andrew Clarke  Added support for WATCOM C/C++ __NT__ macro.
    17-Jun-96  Bob Stout      Added __FLAT__ macros support
    20-Aug-96  Bob Stout      Eliminate Win32 conflicts
========================================================================*/


/* prevent multiple inclusions of this header file */

#ifndef EXTKWORD__H
#define EXTKWORD__H

#include <limits.h>                     /* For INT_MAX, LONG_MAX      */

/*
**  Watcom defines __FLAT__ for 32-bit environments and so will we
*/

#if !defined(__FLAT__) && !defined(__WATCOMC__) && !defined(_MSC_VER)
 #if defined(__GNUC__)
  #define __FLAT__ 1
 #elif defined (_WIN32) || defined(WIN32) || defined(__NT__)
  #define __FLAT__ 1
 #elif defined(__INTSIZE)
  #if (4 == __INTSIZE)
   #define __FLAT__ 1
  #endif
 #elif (defined(__ZTC__) && !defined(__SC__)) || defined(__TURBOC__)
  #if ((INT_MAX != SHRT_MAX) && (SHRT_MAX == 32767))
   #define __FLAT__ 1
  #endif
 #endif
#endif

/*
**  Correct extended keywords syntax
*/

#if defined(__OS2__)        /* EBB: not sure this works for OS/2 1.x */
 #include <os2def.h>
 #define INTERRUPT
 #define HUGE
#elif defined(_WIN32) || defined(WIN32) || defined(__NT__)
 #define WIN32_LEAN_AND_MEAN
 #define NOGDI
 #define NOSERVICE
 #undef INC_OLE1
 #undef INC_OLE2
 #include <windows.h>
 #define INTERRUPT
 #define HUGE
#else /* ! Win 32 or OS/2 */
 #if (defined(__POWERC) || (defined(__TURBOC__) && !defined(__BORLANDC__)) \
       || (defined(__ZTC__) && !defined(__SC__))) && !defined(__FLAT__)
  #define FAR far
  #define NEAR near
  #define PASCAL pascal
  #define CDECL cdecl
  #if (defined(__ZTC__) && !defined(__SC__)) || (defined(__SC__) && \
        (__SC__ < 0x700))
   #define HUGE far
   #define INTERRUPT
  #else
   #define HUGE huge
   #define INTERRUPT interrupt
  #endif
 #else
  #if (defined(__MSDOS__) || defined(MSDOS)) && !defined(__FLAT__)
   #define FAR _far
   #define NEAR _near
   #define HUGE _huge
   #define PASCAL _pascal
   #define CDECL _cdecl
   #define INTERRUPT _interrupt
  #else
   #define FAR
   #define NEAR
   #define HUGE
   #define PASCAL
   #define CDECL
  #endif
 #endif
#endif
```

```
 101 |
 102 | #endif /* EXTKWORD__H */
```

TEXT STATISTICS

```
  2858 characters
   102 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
    62 preprocessor instructions
     0 constants [character]
     0 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    CDECL           75      90      97
    EXTKWORD__H             27      28
    FAR             72      86      93
    HUGE            59      68      78      81      88      95
    INC_OLE1        64
    INC_OLE2        65
    INTERRUPT       58      67      79      82      91
    INT_MAX         46
    MSDOS           85
    NEAR            73      87      94
    NOGDI           62
    NOSERVICE       63
    PASCAL          74      89      96
    SHRT_MAX        46+
    WIN32           39      60
    WIN32_LEAN_AND_MEAN             61
    _MSC_VER        36
    _WIN32          39      60
    __BORLANDC__            70
    __FLAT__        36      38      40      43      47      71      85
    __GNUC__        37
    __INTSIZE       41      42
    __MSDOS__       85
    __NT__          39      60
    __OS2__         56
    __POWERC        70
    __SC__          45      71      76+     77
    __TURBOC__              45      70
    __WATCOMC__             36
    __ZTC__         45      71      76
    _cdecl          90
    _far            86
    _huge           88
    _interrupt              91
    _near           87
    _pascal         89
    cdecl           75
    defined         36+     37      39+     41      45+     56      60+     70+     71+     76+     85+
    far             72      78
    h       30      57      66
    huge            81
    interrupt       82
    limits          30
    near            73
    os2def          57
    pascal          74
    windows         66
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ext_getch()
**
**  A getch() work-alike for use with extended keyboards.
**
**  Parameters: none
**
**  Returns: Extended key code as follows:
**           0->255     Normal key
**           256->511   Numeric pad key or Function key
**           512->767   Cursor pad key or Numeric pad
**                      "duplicate" key (Enter, /, *, -, +)
**
**  Original Copyright 1992 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Revisions:
**  30-Mar-96  Ed Blackman  OS/2 mods
*/

#include <dos.h>
#include <ctype.h>
#include "hilobyte.h"
#include "snipkbio.h"
#include "ext_keys.h"

#define USING_DOS 0     /* Set to 1 to call DOS instead of the BIOS     */

int ext_getch(void)
{
      int key;
#ifdef __OS2__
      extern KBDKEYINFO ki;         /* defined in ISSHIFT.C */
      KBDINFO kb_state;

      kb_state = setkbmode();       /* Change keyboard to binary mode */
      KbdCharIn(&ki, IO_WAIT, 0);   /* Get the key */
      restkbmode(kb_state);         /* restore previous keyboard mode */

      key = (ki.chScan << 8) + ki.chChar;       /* format it into an int */
#else                         /* assume DOS */
      union REGS regs;

 #if USING_DOS
      regs.h.ah = 7;
      intdos(&regs, &regs);
      key = regs.h.al;
      if (0 == key)
      {
            regs.h.ah = 7;
            intdos(&regs, &regs);
            key = (regs.h.al << 8);
      }
 #else
      regs.h.ah = 0x10;
      int86(0x16, &regs, &regs);
      key = regs.x.ax;
 #endif

      switch (LoByte(key))
      {
      case 0:
            key = HiByte(key) + 256;
            break;

      case 0xe0:
            key = HiByte(key) + 512;
            break;

      default:
            if (0xe0 == HiByte(key))
                  key = LoByte(key) + 512;
            else
            {
                  if (ispunct(LoByte(key)) && HiByte(key) > 0x36)
                        key = LoByte(key) + 512;
                  else  key = LoByte(key);
            }
      }
#endif
      return key;
}

int GetExtKey(int *isshift)
{
      int key =  ext_getch();

      *isshift = IsShift();
      return key;
}

#ifdef __OS2__
KBDINFO setkbmode(void)
{
```

```
101 |     USHORT rc;
102 |     KBDINFO kb_state;
103 |
104 |     kb_state.cb = sizeof(kb_state);
105 |     KbdGetStatus(&kb_state, 0);
106 |     kb_state.fsMask &= ~KEYBOARD_ASCII_MODE;
107 |     kb_state.fsMask |= KEYBOARD_BINARY_MODE;
108 |     rc = KbdSetStatus(&kb_state, 0);
109 |
110 | /*  if(rc) printf("KbdSetStatus rc = %04x\n", rc);  */
111 |
112 |     return kb_state;
113 | }
114 |
115 | void restkbmode(KBDINFO kb_state)    /* restore keyboard mode */
116 | {
117 |     USHORT rc;
118 |
119 |     rc = KbdSetStatus(&kb_state, 0);
120 | }
121 | #endif      /* !__OS2__ */
122 |
123 | #ifdef TEST
124 |
125 | #include <stdio.h>
126 |
127 | main()
128 | {
129 |       int key0, key1, shift;
130 |
131 |       puts("Hit keys, Esc twice to stop\n");
132 |       for ( key0 = key1 = 0; !(key1 == Key_ESC && key0 == Key_ESC); )
133 |       {
134 |             key0 = key1;
135 |             key1 = GetExtKey(&shift);
136 |             printf("ext_getch() returned %0Xh, Shift is O%s\n",
137 |                   key1, shift ? "n" : "ff");
138 |       }
139 |       return 0;
140 | }
141 |
142 | #endif /* TEST */
```

TEXT STATISTICS

```
  3416 characters
   142 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    17 preprocessor instructions
     0 constants [character]
     7 constants [string]
    22 constants [numeric]
```

SYMBOL TABLE

```
    GetExtKey       90     135
    HiByte          69      73      77      81
    IO_WAIT         43
    IsShift         94
    KBDINFO         40      99     102     115
    KBDKEYINFO              39
    KEYBOARD_ASCII_MODE            106
    KEYBOARD_BINARY_MODE           107
    KbdCharIn       43
    KbdGetStatus           105
    KbdSetStatus           108     119
    Key_ESC        132+
    LoByte          66      78      81      82      83
    REGS            48
    TEST           123
    USHORT         101     117
    USING_DOS       33      50
    __OS2__         38      98
    ah              51      56      61
    al              53      58
    ax              63
    cb             104
    chChar          46
    chScan          46
    ctype           28
    dos             27
    ext_getch       35      92
    fsMask         106     107
    h       27      28      51      53      56      58      61     125
    int86           62
    intdos          52      57
    ispunct         81
    isshift         90      94
    kb_state        40      42      44     102     104+    105     106     107     108     112     115
           119
    key             37      46      53      54      58      63      66      69+     73+     77      78+
            81+     82+     83+     87      92      95
    key0           129     132+    134
    key1           129     132+    134     135     137
    ki              39      43      46+
    main           127
    printf         136
    puts           131
    rc             101     108     117     119
    regs            48      51      52+     53      56      57+     58      61      62+     63
    restkbmode              44     115
    setkbmode       42      99
    shift          129     135     137
    stdio          125
    x       63
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Keyboard I/O header file.
**
**  ext_getch() returns:
**
**           0->255     Normal key
**           256->511   Numeric pad key or Function key
**           512->767   Cursor pad key or Numeric pad
**                      "duplicate" key (Enter, /, *, -, +)
**
**  GetExtKey() returns same as ext_getch() plus shift key state.
**
**  ext_inkey() returns:
**           EOF if no pending keypress,
**           else same values as ext_getch().
**
**  fast_kbhit() returns non-zero if keypress pending.
**
**  fast_kbflush() clears all pending keypresses.
*/

#ifndef EXT_KEYS__H
#define EXT_KEYS__H

int  ext_getch(void);                           /* Ext_Keys.C     */
int  GetExtKey(int *isshift);                   /* Ext_Keys.C     */
int  fast_kbhit_dos(void);                      /* Faskbhit.C     */
void fast_kbflush_dos(void);                    /* Faskbhit.C     */
int  fast_kbhit_os2(void);                      /* Faskbhit.C     */
void fast_kbflush_os2(void);                    /* Faskbhit.C     */
int  ext_inkey(void);                           /* Faskbhit.C     */

#if defined(__OS2__)
 #define fast_kbhit()   fast_kbhit_os2()
 #define fast_kbflush() fast_kbflush_os2()
#else /* Assume DOS */
 #define fast_kbhit()   fast_kbhit_dos()
 #define fast_kbflush() fast_kbflush_dos()
#endif

#define Key_ESC         0x1b        /* ASCII codes                */
#define Key_ENTER       '\r'
#define Key_TAB         '\t'
#define Key_BACKSPC     '\b'
#define Key_NL          '\n'
#define Key_LFEED       '\n'
#define Key_FFEED       '\f'

#define Key_F1          0x13b       /* Function keys              */
#define Key_F2          0x13c
#define Key_F3          0x13d
#define Key_F4          0x13e
#define Key_F5          0x13f
#define Key_F6          0x140
#define Key_F7          0x141
#define Key_F8          0x142
#define Key_F9          0x143
#define Key_F10         0x144
#define Key_F11         0x185
#define Key_F12         0x186

#define Key_CF1         0x15e       /* Ctrl-Function keys         */
#define Key_CF2         0x15f
#define Key_CF3         0x160
#define Key_CF4         0x161
#define Key_CF5         0x162
#define Key_CF6         0x163
#define Key_CF7         0x164
#define Key_CF8         0x165
#define Key_CF9         0x166
#define Key_CF10        0x167
#define Key_CF11        0x189
#define Key_CF12        0x18a

#define Key_SF1         0x154       /* Shift-Function keys        */
#define Key_SF2         0x155
#define Key_SF3         0x156
#define Key_SF4         0x157
#define Key_SF5         0x158
#define Key_SF6         0x159
#define Key_SF7         0x15a
#define Key_SF8         0x15b
#define Key_SF9         0x15c
#define Key_SF10        0x15d
#define Key_SF11        0x187
#define Key_SF12        0x188

#define Key_AF1         0x168       /* Alt-Function keys          */
#define Key_AF2         0x169
#define Key_AF3         0x16a
#define Key_AF4         0x16b
#define Key_AF5         0x16c
#define Key_AF6         0x16d
#define Key_AF7         0x16e
#define Key_AF8         0x16f
#define Key_AF9         0x170
#define Key_AF10        0x171
#define Key_AF11        0x18b
```

```
#define Key_AF12        0x18c

#define Key_INS         0x152       /* Numeric pad keys           */
#define Key_DEL         0x153
#define Key_HOME        0x147
#define Key_END         0x14f
#define Key_PGUP        0x149
#define Key_PGDN        0x151
#define Key_UPARROW     0x148
#define Key_DNARROW     0x150
#define Key_LTARROW     0x14b
#define Key_RTARROW     0x14d
#define Key_PADMIDDLE   0x14c

#define Key_PADEQ       0x3d        /* Numeric pad grey keys      */
#define Key_PADPLUS     0x22b
#define Key_PADMINUS    0x22d
#define Key_PADASTERISK 0x22a
#define Key_PADSLASH    0x22f
#define Key_PADENTER    0x20d

#define Key_CEND        0x175       /* Ctrl-Numeric pad keys      */
#define Key_CDNARROW    0x191
#define Key_CPGDN       0x176
#define Key_CLTARROW    0x173
#define Key_CPADMIDDLE  0x18f
#define Key_CRTARROW    0x174
#define Key_CHOME       0x177
#define Key_CUPARROW    0x18d
#define Key_CPGUP       0x184
#define Key_CINS        0x192
#define Key_CDEL        0x193

#define Key_PINS        0x252       /* Cursor pad keys            */
#define Key_PDEL        0x253
#define Key_PHOME       0x247
#define Key_PEND        0x24f
#define Key_PPGUP       0x249
#define Key_PPGDN       0x251
#define Key_PUPARROW    0x248
#define Key_PDNARROW    0x250
#define Key_PLTARROW    0x24b
#define Key_PRTARROW    0x24d

#define Key_CPEND       0x275       /* Ctrl-Cursor pad keys       */
#define Key_CPDNARROW   0x291
#define Key_CPPGDN      0x276
#define Key_CPLTARROW   0x273
#define Key_CPRTARROW   0x274
#define Key_CPHOME      0x277
#define Key_CPUPARROW   0x28d
#define Key_CPPGUP      0x284
#define Key_CPINS       0x292
#define Key_CPDEL       0x293

#define Key_ALTPSLASH   0x1a4       /* Alt-numeric pad grey keys  */
#define Key_ALTPASTRSK  0x137
#define Key_ALTPMINUS   0x14a
#define Key_ALTPPLUS    0x14e
#define Key_ALTPEQUALS  0x183
#define Key_ALTPENTER   0x1a6

#define Key_ALTBACKSPC  0x10e       /* Special PC keyboard keys   */
#define Key_CTRLBACKSPC 0x7f
#define Key_SHIFTTAB    0x10f
#define Key_CTRLTAB     0x194
#define Key_ALTESC      0x101

#define Key_ALT1        0x178       /* Alt keys - number row      */
#define Key_ALT2        0x179
#define Key_ALT3        0x17a
#define Key_ALT4        0x17b
#define Key_ALT5        0x17c
#define Key_ALT6        0x17d
#define Key_ALT7        0x17e
#define Key_ALT8        0x17f
#define Key_ALT9        0x180
#define Key_ALT0        0x181
#define Key_ALTMINUS    0x182
#define Key_ALTEQUALS   0x183

#define Key_ALTQ        0x110       /* Alt keys - top alpha row   */
#define Key_ALTW        0x111
#define Key_ALTE        0x112
#define Key_ALTR        0x113
#define Key_ALTT        0x114
#define Key_ALTY        0x115
#define Key_ALTU        0x116
#define Key_ALTI        0x117
#define Key_ALTO        0x118
#define Key_ALTP        0x119
#define Key_ALTLBRACE   0x11a
#define Key_ALTRBRACE   0x11b

#define Key_ALTA        0x11e       /* Alt keys - mid alpha row   */
#define Key_ALTS        0x11F
#define Key_ALTD        0x120
#define Key_ALTF        0x121
#define Key_ALTG        0x122
#define Key_ALTH        0x123
```

```
201 | #define Key_ALTJ        0x124
202 | #define Key_ALTK        0x125
203 | #define Key_ALTL        0x126
204 | #define Key_ALTCOLON    0x127
205 | #define Key_ALTQUOTE    0x128
206 | #define Key_ALTENTER    0x11c
207 |
208 | #define Key_ALTZ        0x12c       /* Alt keys - lower alpha row */
209 | #define Key_ALTX        0x12d
210 | #define Key_ALTC        0x12e
211 | #define Key_ALTV        0x12f
212 | #define Key_ALTB        0x130
213 | #define Key_ALTN        0x131
214 | #define Key_ALTM        0x132
215 | #define Key_ALTCOMMA    0x133
216 | #define Key_ALTPERIOD   0x134
217 | #define Key_ALTSLASH    0x135
218 | #define Key_ALTBSLASH   0x12b
219 | #define Key_ALTTILDE    0x129
220 |
221 | #endif /* EXT_KEYS__H */
```

```
TEXT STATISTICS

  7094 characters
   221 lines

LEXICAL STATISTICS

    27 comments [std-C]
     0 comments [C++]
   172 preprocessor instructions
     6 constants [character]
     0 constants [string]
   156 constants [numeric]

SYMBOL TABLE

    EXT_KEYS__H             24      25
    GetExtKey       28
    Key_AF1         90
    Key_AF10        99
    Key_AF11       100
    Key_AF12       101
    Key_AF2         91
    Key_AF3         92
    Key_AF4         93
    Key_AF5         94
    Key_AF6         95
    Key_AF7         96
    Key_AF8         97
    Key_AF9         98
    Key_ALT0       178
    Key_ALT1       169
    Key_ALT2       170
    Key_ALT3       171
    Key_ALT4       172
    Key_ALT5       173
    Key_ALT6       174
    Key_ALT7       175
    Key_ALT8       176
    Key_ALT9       177
    Key_ALTA       195
    Key_ALTB       212
    Key_ALTBACKSPC          163
    Key_ALTBSLASH           218
    Key_ALTC       210
    Key_ALTCOLON            204
    Key_ALTCOMMA            215
    Key_ALTD       197
    Key_ALTE       184
    Key_ALTENTER            206
    Key_ALTEQUALS           180
    Key_ALTESC              167
    Key_ALTF       198
    Key_ALTG       199
    Key_ALTH       200
    Key_ALTI       189
    Key_ALTJ       201
    Key_ALTK       202
    Key_ALTL       203
    Key_ALTLBRACE           192
    Key_ALTM       214
    Key_ALTMINUS            179
    Key_ALTN       213
    Key_ALTO       190
    Key_ALTP       191
    Key_ALTPASTRSK          157
    Key_ALTPENTER           161
    Key_ALTPEQUALS          160
    Key_ALTPERIOD           216
    Key_ALTPMINUS           158
    Key_ALTPPLUS            159
    Key_ALTPSLASH           156
    Key_ALTQ       182
    Key_ALTQUOTE            205
    Key_ALTR       185
    Key_ALTRBRACE           193
    Key_ALTS       196
    Key_ALTSLASH            217
    Key_ALTT       186
    Key_ALTTILDE            219
    Key_ALTU       188
    Key_ALTV       211
    Key_ALTW       183
    Key_ALTX       209
    Key_ALTY       187
    Key_ALTZ       208
    Key_BACKSPC              46
    Key_CDEL       132
    Key_CDNARROW            123
    Key_CEND       122
    Key_CF1         64
    Key_CF10        73
    Key_CF11        74
    Key_CF12        75
    Key_CF2         65
    Key_CF3         66
    Key_CF4         67
    Key_CF5         68
    Key_CF6         69
    Key_CF7         70
```

```
Key_CF8          71
Key_CF9          72
Key_CHOME       128
Key_CINS        131
Key_CLTARROW             125
Key_CPADMIDDLE           126
Key_CPDEL       154
Key_CPDNARROW            146
Key_CPEND       145
Key_CPGDN       124
Key_CPGUP       130
Key_CPHOME               150
Key_CPINS       153
Key_CPLTARROW            148
Key_CPPGDN               147
Key_CPPGUP               152
Key_CPRTARROW            149
Key_CPUPARROW            151
Key_CRTARROW             127
Key_CTRLBACKSPC          164
Key_CTRLTAB              166
Key_CUPARROW             129
Key_DEL         104
Key_DNARROW              110
Key_END         106
Key_ENTER        44
Key_ESC          43
Key_F1           51
Key_F10          60
Key_F11          61
Key_F12          62
Key_F2           52
Key_F3           53
Key_F4           54
Key_F5           55
Key_F6           56
Key_F7           57
Key_F8           58
Key_F9           59
Key_FFEED        49
Key_HOME        105
Key_INS         103
Key_LFEED        48
Key_LTARROW              111
Key_NL           47
Key_PADASTERISK          118
Key_PADENTER             120
Key_PADEQ       115
Key_PADMIDDLE            113
Key_PADMINUS             117
Key_PADPLUS              116
Key_PADSLASH             119
Key_PDEL        135
Key_PDNARROW             141
Key_PEND        137
Key_PGDN        108
Key_PGUP        107
Key_PHOME       136
Key_PINS        134
Key_PLTARROW             142
Key_PPGDN       139
Key_PPGUP       138
Key_PRTARROW             143
Key_PUPARROW             140
Key_RTARROW              112
Key_SF1          77
Key_SF10         86
Key_SF11         87
Key_SF12         88
Key_SF2          78
Key_SF3          79
Key_SF4          80
Key_SF5          81
Key_SF6          82
Key_SF7          83
Key_SF8          84
Key_SF9          85
Key_SHIFTTAB             165
Key_TAB          45
Key_UPARROW              109
__OS2__          35
defined          35
ext_getch        27
ext_inkey        33
fast_kbflush              37      40
fast_kbflush_dos          30      40
fast_kbflush_os2          32      37
fast_kbhit                36      39
fast_kbhit_dos            29      39
fast_kbhit_os2            31      36
isshift          28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** factor.c -- print prime factorization of a number
** Ray Gardner -- 1985 -- public domain
** Modified Feb. 1989 by Thad Smith > public domain
**
** This version takes numbers up to the limits of double precision.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int prevfact = 0;
void factor (double);
void show (double, int);

main (int argc, char *argv[])
{
      while ( --argc )
            factor(atof(*++argv));
      return 0;
}

void factor (double n)
{
      double d;
      int k;

      prevfact = 0;

      d = n+1;     /* test for roundoff error */
      if (n+3 != d+2)
      {
            printf("%0.0f is too large to process.\n", n);
            return;
      }
      if (fmod(n,1.) != 0.0)
      {
            printf("%f is not an integer.\n",n);
            return;
      }
      printf("%0.0f  ",n);
      if ( n < 2. )
      {
            printf("is less than 2.\n");
            return;
      }
      else if ( n > 2. )
      {
            d = 2;
            for ( k = 0; fmod(n,d) == 0.0; k++ )
                  n /= d;
            if ( k )
                  show(d,k);
            for ( d = 3; d * d <= n; d += 2 )
            {
                  for ( k = 0; fmod(n,d) == 0.0; k++ )
                        n /= d;
                  if ( k )
                        show(d,k);
            }
      }
      if ( n > 1 )
      {
            if ( ! prevfact )
                  printf(" is prime");
            else  show(n,1);
      }
      printf("\n");
}

void show (double d, int k)
{
      if ( prevfact )
            printf(" * ");
      else  printf(" = ");
      prevfact++;
      printf("%0.0f",d);
      if ( k > 1 )
            printf("^%d",k);
}
```

TEXT STATISTICS

```
  1824 characters
    83 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    10 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

argc		19	21									
argv		19	22									
atof		22										
d	28	33	34	52	53	54	56	57+	59	60	62	74
	80											
factor		16	22	26								
fmod		39	53	59								
h	11	12	13									
k	29	53+	55	56	59+	61	62	74	81	82		
main		19										
math		13										
n	26	33	34	36	39	41	44	45	50	53	54	57
	59	60	65	69								
prevfact		15	31	67	76	79						
printf		36	41	44	47	68	71	77	78	80	82	
show		17	56	62	69	74						
stdio		11										
stdlib		12										

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FACTORYL.C
**
**  Original Copyright 1992 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Uses DBLROUND.C, also in SNIPPETS
*/

#include <float.h>
#include "snipmath.h"

#define dfrac(x) ((x)-dround(x))

#define SQRT2PI sqrt(2 * PI)
#define ONESIXTH (1.0/6.0)

/*
**  log10factorial()
**
**  Returns the logarithm (base 10) of the factorial of a given number.
**  The logarithm is returned since this allows working with extremely
**  large values which would otherwise overflow the F.P. range.
**
**  Parameters: 1 - Number whose factorial to return.
**
**  Returns: log10() of the passed value, -1.0 if error
**
**  Limitations: Cannot return 0! since log(0) is undefined.
*/

double log10factorial(double N)
{
      double dummy;

      if ((N < 1.0) || (0.0 != modf(N, &dummy)))
            return -1.0;
      if (N < 40)                         /* Small, explicitly compute  */
      {
            int i;
            double f;

            if (0.0 == N)
                  return N;
            for (i = 1, f = 1.0; i <= (int)N; ++i)
                  f *= i;
            return log10(f);
      }
      else                                /* Large, use approximation   */
      {
            return log10(SQRT2PI)+((N + 0.5) *
                  (log10(sqrt(N * N + N + ONESIXTH) / exp(1.0))));
      }
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      double f, lf;
      char *dummy;

      while (--argc)
      {
            f  = strtod((const char *)(*(++argv)), &dummy);
            if (0.0 == f)
            {
                  puts("0! = 0");
                  continue;
            }
            if (-1.0 == (lf = log10factorial(f)))
            {
                  printf(">>> ERROR: %g! is not a valid expression\n", f);
                  continue;
            }
            if (171.0 > f)
                  printf("%.14g! = %.14g\n", f, pow(10.0, lf));
            else
            {
                  printf("%.14g! = %.14ge+%ld\n", f,
                        pow(10.0, dfrac(lf)), (long)dround(lf));
            }
      }
      lf = log10C(1000000L,750000L);
      printf("\nJust to dazzle with you with big numbers:\n"
            "C(1000000,750000) = %.14ge+%ld\n",
            pow(10.0, dfrac(lf)), (long)dround(lf));
      lf = log10P(1000000L,750000L);
      printf("\n...once more:\n"
            "P(1000000,750000) = %.14ge+%ld\n",
            pow(10.0, dfrac(lf)), (long)dround(lf));
```

```
 101 |       return EXIT_SUCCESS;
 102 | }
 103 |
 104 | #endif /* TEST */


TEXT STATISTICS

  2875 characters
   104 lines

LEXICAL STATISTICS

     6 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     9 constants [string]
    23 constants [numeric]

SYMBOL TABLE

    EXIT_SUCCESS           101
    N       38      42+     44      49      50      51      57      58+
    ONESIXTH        22      58
    PI              21
    SQRT2PI         21      57
    TEST            62
    argc            67      72
    argv            67      74
    dfrac           19      90      96     100
    dround          19      90      96     100
    dummy           40      42      70      74
    exp             58
    f       47      51      52      53      69      74      75      80      82      85      86      89
    h       16      64      65
    i       46      51+     52
    lf              69      80      86      90+     93      96+     97     100+
    log10           53      57      58
    log10C          93
    log10P          97
    log10factorial          38      80
    main            67
    modf            42
    pow             86      90      96     100
    printf          82      86      89      94      98
    puts            77
    sqrt            21      58
    stdio           64
    stdlib          65
    strtod          74
    x       19+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  by David Goodenough & Bob Stout
**
**  Revisions:
**  30-Mar-96  Ed Blackman  OS/2 mods
*/

#include <stdio.h>
#include <dos.h>
#include "snipkbio.h"
#include "extkword.h"
#include "ext_keys.h"
#include "mk_fp.h"

#if defined(__OS2__)

/* 30-Mar-96 - EBB:  it really isn't good to poll the keyboard in OS/2.
** A better design would be to call KbdCharIn(..., IO_WAIT, ...) in a
** separate thread, and sending a message via some form of IPC to the main
** thread when a key is available.  This code is intended to be more of an
** example of how to use the API, rather than something you would copy into
** production code.
*/

int fast_kbhit_os2(void)
{
    extern KBDKEYINFO ki;         /* defined in ISSHIFT.C */

    KbdPeek(&ki, 0);              /* peek in the keyboard buffer */
    DosSleep(1);                  /* give up the rest of the time slice */

    return (ki.fbStatus & (1 << 6));  /* only return true if key is 'final' */
}

void fast_kbflush_os2(void)
{
    KbdFlushBuffer(0);
}

#else   /* assume DOS */
 #define biosseg 0x40

 #define HEAD (*((unsigned short FAR *)MK_FP(biosseg, 0x1a)))
 #define TAIL (*((unsigned short FAR *)MK_FP(biosseg, 0x1c)))

/*
**  Detect a pending keypress
*/

int fast_kbhit_dos(void)
{
      int retval;

      disable();
      retval = HEAD - TAIL;
      enable();
      return retval;
}

/*
**  Clear the keyboard buffer
*/

void fast_kbflush_dos(void)
{
      disable();
      HEAD = TAIL;
      enable();
}
#endif  /* !__OS2__ */

/*
**  Enhanced work-alike for BASIC's INKEY$ function
*/

int ext_inkey(void)
{
      if (!fast_kbhit())
            return EOF;
      else  return ext_getch();
}

#ifdef TEST

#include <conio.h>
#include <time.h>

main()
{
      clock_t Wait;
      int key;

      puts("Hit some keys while I kill some time...");
      Wait = clock();
      while (2 > ((clock() - Wait) / CLK_TCK))
            ;

      puts("OK, stop hitting keys while I flush the keyboard...");
```

```
 101 |       Wait = clock();
 102 |       while (2 > ((clock() - Wait) / CLK_TCK))
 103 |             ;
 104 |
 105 |       fast_kbflush();
 106 |       puts("Optionally, hit some key you didn't hit before...");
 107 |       Wait = clock();
 108 |       while (2 > ((clock() - Wait) / CLK_TCK))
 109 |             ;
 110 |       if (EOF == (key = ext_inkey()))
 111 |             puts("You didn't hit anything");
 112 |       else  printf("You hit extended keycode 0x%04X\n", key);
 113 |
 114 |       puts("OK, now hit some other key you didn't hit before...");
 115 |
 116 |       while (!fast_kbhit())
 117 |             ;
 118 |       printf("You hit extended keycode 0x%04X\n", ext_getch());
 119 |       return 0;
 120 | }
 121 |
 122 | #endif /* TEST */
```

TEXT STATISTICS

```
  2740 characters
   122 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
    11 constants [string]
    12 constants [numeric]
```

SYMBOL TABLE

```
    CLK_TCK         97     102     108
    DosSleep        32
    EOF             81     110
    FAR             45      46
    HEAD            45      57      69
    KBDKEYINFO              29
    KbdFlushBuffer          39
    KbdPeek         31
    MK_FP           45      46
    TAIL            46      57      69
    TEST            85
    Wait            92      96      97     101     102     107     108
    __OS2__         17
    biosseg         43      45      46
    clock           96      97     101     102     107     108
    clock_t         92
    conio           87
    defined         17
    disable         56      68
    dos             11
    enable          58      70
    ext_getch       82     118
    ext_inkey       78     110
    fast_kbflush           105
    fast_kbflush_dos        66
    fast_kbflush_os2        37
    fast_kbhit              80     116
    fast_kbhit_dos          52
    fast_kbhit_os2          27
    fbStatus        34
    h       10      11      87      88
    key             93     110     112
    ki              29      31      34
    main            90
    printf         112     118
    puts            95     100     106     111     114
    retval          54      57      59
    stdio           10
    time            88
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Find out how many more files can be fopen'ed
**
**  public domain demo by Bob Stout
*/

#include "dosfiles.h"

#if defined(__TURBOC__)
 #define STREAM_BUF     _streams
 #define FCNT           FOPEN_MAX
 #define FLAG           flags
#elif defined(__WATCOMC__)
 #define STREAM_BUF     __iob
 #define FCNT           _NFILES
 #define FLAG           _flag
#else /* MSC, ZTC++ */
 #define STREAM_BUF     _iob
 #define FCNT           _NFILE
 #define FLAG           _flag
#endif

int favail(void)
{
      int i, count;

      for (i = count = 0; i < FCNT; ++i)
      {
            if (0 == STREAM_BUF[i].FLAG)
                  ++count;
      }
      return count;
}

#ifdef TEST

main()
{
      char *fname = "A$$$$$$$.$$$";
      FILE *fp;

      do
      {
            int i = favail();

            printf("You can fopen %d new file%s\n", i, &"s"[i == 1]);
            fp = fopen(fname, "w");
            *fname += 1;
      } while (fp);

      do
      {
            printf("removing %s\n", fname);
            remove(fname);
      } while ('A' <= --(*fname)) ;
      return 0;
}

#endif /*TEST */
```

```
TEXT STATISTICS

  1205 characters
    61 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     1 constants [character]
     6 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    FCNT            13      17      21      29
    FILE            42
    FLAG            14      18      22      31
    FOPEN_MAX       13
    STREAM_BUF              12      16      20      31
    TEST            37
    _NFILE          21
    _NFILES         17
    __TURBOC__              11
    __WATCOMC__             15
    __iob           16
    _flag           18      22
    _iob            20
    _streams        12
    count           27      29      32      34
    defined         11      15
    favail          25      46
    flags           14
    fname           41      49      50      55      56      57
    fopen           49
    fp              42      49      51
    i       27      29+     31      46      48+
    main            39
    printf          48      55
    remove          56
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Routines for truncating DOS files associated by streams.
**
** Author   : Eric Coolman, Simple Minded Software
**            modified for SNIPPETS by Bob Stout
**
** Released to public domain on June 23, 1994.  USE AT OWN RISK.
**
** Notes:     fChSize works the same with FILE pointers as CHSIZE() does for
**            file handles.
**
**            All routines return the same values as CHSIZE().
*/


#include <stdio.h>                  /* fileno(), ftell(), FILE *        */
#include <io.h>                     /* chsize()                         */
#include "errors.h"                 /* cant()                           */

#if defined(__ZTC__) && !defined(__SC__)
 #include "sniptype.h"              /* Error_, Success_                 */

 int chsize(int fd, long posn)
 {
       char dummy;

       if (-1L == lseek(fd, posn, SEEK_SET))
             return Error_;
       if (-1 == write(fd, &dummy, 0))
             return Error_;
       else    return Success_;
 }
#endif

/*
** Expand or reduce the size of a file
*/

int fChSize(FILE *stream, long size)
{
      return(chsize(fileno(stream), size));
}

/*
** Truncates the file at the current offset of the FILE pointer
*/

int fTrunc(FILE *stream)
{
      return(fChSize(stream, ftell(stream)));
}

/*
** Clears contents of stream (zero bytes it)
*/

int fStub(FILE *stream)
{
      return fChSize(stream, 0L);
}


#ifdef TEST

#include <stdlib.h>     /* atol()         */
#include <ctype.h>      /* toupper()      */

void usage(void)
{
      puts("Usage: FCHSIZE { C | S } filename [new_length]\n");
      puts("Where: C  means change the file size, "
           "new size given by new_length");
      puts("       S  means truncate the file to zero bytes");
      exit(EXIT_FAILURE);
}

int main(int argc, char *argv[])
{
      FILE *testfile;
      long posn;

      if (argc < 3)
            usage();

      testfile = cant(argv[2], "r+b");

      switch (toupper(argv[1][0]))
      {
      case 'C':
            posn    = atol(argv[3]);
            printf("fChSize(%s, %ld) returned %d\n", argv[1], posn,
                   fChSize(testfile, posn));
            break;

      case 'S':
            printf("fStub(%s) returned %d\n", argv[1], fStub(testfile));
            break;
```

```
 101 |       default:
 102 |             usage();
 103 |       }
 104 |
 105 |       fclose(testfile);
 106 |       return EXIT_SUCCESS;
 107 | }
 108 |
 109 | #endif /* TEST */
```

TEXT STATISTICS

```
  2526 characters
   109 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     2 constants [character]
     9 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            76
    EXIT_SUCCESS           106
    Error_          30      32
    FILE            41      50      59      81
    SEEK_SET        29
    Success_        33
    TEST            65
    __SC__          22
    __ZTC__         22
    argc            79      84
    argv            79      87      89      92      93      98
    atol            92
    cant            87
    chsize          25      43
    ctype           68
    defined         22+
    dummy           27      31
    exit            76
    fChSize         41      52      61      94
    fStub           59      98
    fTrunc          50
    fclose         105
    fd              25      29      31
    fileno          43
    ftell           52
    h       18      19      67      68
    io              19
    lseek           29
    main            79
    posn            25      29      82      92      93      94
    printf          93      98
    puts            72      73      75
    size            41      43
    stdio           18
    stdlib          67
    stream          41      43      50      52+     59      61
    testfile        81      87      94      98     105
    toupper         89
    usage           70      85     102
    write           31
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FCOMPARE.C - Compare 2 files
**
**  public domain demo by Bob Stout
*/

#include <stdlib.h>
#include <string.h>
#include "snipfile.h"
#include "sniptype.h"

#define BUFSIZE 16384
static char buf[2][BUFSIZE];

int fcompare(const char *fname1, const char *fname2)
{
      FILE *f1, *f2;
      int retval = Success_;

      if (NULL == (f1 = fopen(fname1, "rb")))
            return Error_;
      if (NULL != (f2 = fopen(fname2, "rb")))
      {
            size_t size1, size2;

            fseek(f1, 0L, SEEK_END);
            fseek(f2, 0L, SEEK_END);

            if (ftell(f1) != ftell(f2))
                  retval = !Success_;
            else
            {
                  rewind(f1);
                  rewind(f2);
                  do
                  {
                        size1 = fread(buf[0], 1, BUFSIZE, f1);
                        size2 = fread(buf[1], 1, BUFSIZE, f2);
                        if (0 == (size1 | size2))
                              break;
                        if ((size1 != size2) || memcmp(buf[0], buf[1], size1))
                        {
                              retval = !Success_;
                              break;
                        }
                  } while (size1 && size2);
            }
            fclose(f2);
      }
      else  retval = Error_;
      fclose(f1);
      return retval;
}

#ifdef TEST

int main(int argc, char *argv[])
{
      if (3 > argc)
      {
            puts("Usage: FCOMPARE file1 file2");
            return 1;
      }
      printf("fcompare(%s, %s) returned %d\n", argv[1], argv[2],
            fcompare(argv[1], argv[2]));
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1771 characters
    71 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     6 constants [string]
    18 constants [numeric]

SYMBOL TABLE

    BUFSIZE          14      15      39      40
    Error_           23      52
    FILE             19
    NULL             22      24
    SEEK_END         28      29
    Success_         20      32      45
    TEST             57
    argc             59      61
    argv             59      66+     67+
    buf              15      39      40      43+
    f1               19      22      28      31      35      39      53
    f2               19      24      29      31      36      40      50
    fclose           50      53
    fcompare         17      67
    fname1           17      22
    fname2           17      24
    fopen            22      24
    fread            39      40
    fseek            28      29
    ftell            31+
    h        9       10
    main             59
    memcmp           43
    printf           66
    puts             63
    retval           20      32      45      52      54
    rewind           35      36
    size1            26      39      41      43+     48
    size2            26      40      41      43      48
    size_t           26
    stdlib            9
    string           10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Public domain by Bob Stout
*/

#if !defined(__TURBOC__) && !defined(__SC__)
 #include "ftime.h"
#endif
#include <io.h>
#include <fcntl.h>
#include "ftime.h"
#include "scaldate.h"

/*
**  getfdate() - Return DOS file (handle) date as a SNIPPETS scalar date
*/

int getfdate (int handle, long *date)
{
      static struct ftime ftimep;
      int retval = 0;

      if (0 == (retval = getftime(handle, &ftimep)))
      {
            *date = ymd_to_scalar(ftimep.ft_year + 1980, ftimep.ft_month,
                  ftimep.ft_day);
      }
      return retval;
}

/*
**  getdatef() - Return DOS file (filename) date as a SNIPPETS scalar date
*/

int getdatef (char *fname, long *date)
{
      int fh;

      if (-1 != (fh = open(fname, O_RDONLY, 0)))
      {
            int retval;

            retval = getfdate(fh, date);
            close(fh);
            return retval;
      }
      else  return fh;
}

#ifdef TEST

#include <stdio.h>

main(int argc, char *argv[])
{
      long date;

      while (--argc)
      {
            if (0 == getdatef(*++argv, &date))
            {
                  unsigned yr, mo, dy;

                  scalar_to_ymd(date, &yr, &mo, &dy);
                  printf("%-15s: %2d/%02d/%04d\n", *argv, mo, dy, yr);
            }
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1420 characters
    72 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     4 constants [string]
     7 constants [numeric]

SYMBOL TABLE

    O_RDONLY        40
    TEST            51
    __SC__           7
    __TURBOC__               7
    argc            55      59
    argv            55      61      66
    close           45
    date            19      26      36      44      57      61      65
    defined          7+
    dy              63      65      66
    fcntl           11
    fh              38      40      44      45      48
    fname           36      40
    ft_day          27
    ft_month        26
    ft_year         26
    ftime           21
    ftimep          21      24      26+     27
    getdatef        36      61
    getfdate        19      44
    getftime        24
    h       10      11      53
    handle          19      24
    io              10
    main            55
    mo              63      65      66
    open            40
    printf          66
    retval          22      24      29      42      44      46
    scalar_to_ymd           65
    stdio           53
    ymd_to_scalar           26
    yr              63      65      66
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FERRORF.C - Functions for formatted error messages
*/

/*
**  ferrorf()
**
**  Prints error message with printf() formatting syntax, then a colon,
**  then a message corresponding to the value of errno, then a newline.
**  Output is to filehandle.
**
**  Public Domain by Mark R. Devlin, free usage is permitted.
*/

#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "errors.h"

int ferrorf(FILE *filehandle, const char *format, ...)
{
      int vfp, fp;
      va_list vargs;

      vfp = fp = 0;
      va_start(vargs, format);
      vfp = vfprintf(filehandle, format, vargs);
      va_end(vargs);
      fp = fprintf(filehandle, ": %s\n", sys_errlist[errno]);
      return ((vfp==EOF || fp==EOF) ? EOF : (vfp+fp));
}

/*
**  cant() - An fopen() replacement with error trapping
**
**  Call just as you would fopen(), but make sure your exit functions are
**  registered with atexit().
**
**  public domain by Bob Stout
*/

FILE *cant(char *fname, char *fmode)
{
      FILE *fp;

      if (NULL == (fp = fopen(fname, fmode)))
      {
            ferrorf(stderr, "Can't open %s", fname);
            exit(EXIT_FAILURE);
      }
      return fp;
}

#ifdef TEST

main()
{
      char badname[] = "????????.???";
      FILE *fp;

      fp = cant(badname, "r");
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1429 characters
    67 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     5 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    EOF             32+
    EXIT_FAILURE            51
    FILE            22      44      46      61
    NULL            48
    TEST            56
    badname         60      63
    cant            44      63
    errno           31
    exit            51
    ferrorf         22      50
    filehandle              22      29      31
    fmode           44      48
    fname           44      48      50
    fopen           48
    format          22      28      29
    fp              24      27      31      32+     46      48      53      61      63
    fprintf         31
    h       17      18      19
    main            58
    stdarg          18
    stderr          50
    stdlib          17
    string          19
    sys_errlist             31
    va_end          30
    va_list         25
    va_start        28
    vargs           25      28      29      30
    vfp             24      27      29      32+
    vfprintf        29
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 |     This program demonstrates how to calculate any arbitrary term
  5 |     of the Fibonacci sequence using phi (the golden number) and
  6 |     eighteenth century mathematician A. de Moivre's formula.
  7 |
  8 |     written on Fri  04-05-1996  by Ed Beroset
  9 |         and released to the public domain by the author
 10 |
 11 |     modified for SNIPPETS 01-Sep-1996 by Bob Stout
 12 | */
 13 |
 14 | #include "snipmath.h"                          /* includes math.h      */
 15 |
 16 | double fibo(unsigned short term)
 17 | {
 18 |       const double k = 1/sqrt(5.0);            /* a handy constant     */
 19 |       double x;                                 /* the Fibonacci term   */
 20 |
 21 |       /*
 22 |       **  this is de Moivre's formula using phi, the golden number (defined
 23 |       **  in snipmath.h), and as simplified by Donald Knuth
 24 |       */
 25 |
 26 |       x = k * pow(PHI, term);
 27 |
 28 |       /*
 29 |       **  this is to compensate for computer error
 30 |       */
 31 |
 32 |       return dround(x);
 33 | }
 34 |
 35 | #ifdef TEST
 36 |
 37 | #include <stdio.h>
 38 |
 39 | int main()
 40 | {
 41 |       unsigned i;                               /* which term?          */
 42 |
 43 |       for (i = 1; i < 84; i++)                  /* limit of precision   */
 44 |             printf("%2d: %20.0f\n", i, fibo(i));
 45 |       return 0;
 46 | }
 47 |
 48 | #endif /* TEST */
```

TEXT STATISTICS

```
  1271 characters
    48 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     2 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    PHI             26
    TEST            35
    dround          32
    fibo            16      44
    h       37
    i       41      43+     44+
    k       18      26
    main            39
    pow             26
    printf          44
    sqrt            18
    stdio           37
    term            16      26
    x       19      26      32
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FILCOUNT.C - counts directories and /or files in a directory
**
**  public domain demo by Bob Stout
*/

#include <stdio.h>
#include <string.h>
#include "sniptype.h"
#include "dirport.h"

unsigned DirCount = 0, FileCount = 0;

/*
**  Arguments: 1 - directory to search
**             2 - search subdirectories: True_ or False_
*/

void do_dir(char *path, int recurse_flag)
{
      char search[67], new[67];
      DOSFileData ff;

      strcpy(search, path);
      if ('\\' != LAST_CHAR(search))
            strcat(search, "\\");
      strcat(search, "*.*");
      if (Success_ == FIND_FIRST(search, _A_ANY, &ff)) do
      {
            if ('.' == *ff_name(&ff))
                  continue;
            if (ff_attr(&ff) & _A_SUBDIR)
            {
                  DirCount++;
                  if (recurse_flag)
                  {
                        strcpy(new, path);
                        if  ('\\' != LAST_CHAR(new))
                              strcat(new, "\\");
                        strcat(new, ff_name(&ff));
                        do_dir(new, recurse_flag);
                  }
            }
            else  FileCount++;
      } while (Success_ == FIND_NEXT(&ff));
}

/*
**  Simple recursive file/directory counter
**
**  Usage: FILCOUNT [path_name] [{Y | N}]
**
**  Notes: 1. If a path name isn't specified, the current directory is assumed
**         2. Default recursion flag is False_
**         3. Path must be specified in order to specify the recursion flag
*/

main(int argc, char *argv[])
{
      char *Dir =".";
      Boolean_T recurse = False_;

      if (1 < argc)
            Dir = argv[1];
      if (2 < argc)
            recurse = (NULL != strchr("Yy", *argv[2]));
      do_dir(Dir, recurse);
      printf("Counted: %d Directories and %d Files\n",
            DirCount, FileCount);
      return 0;
}
```

```
TEXT STATISTICS

  1959 characters
    73 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     3 constants [character]
     8 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    Boolean_T       63
    DOSFileData             24
    Dir             62      66      69
    DirCount        14      36      71
    FIND_FIRST              30
    FIND_NEXT       47
    False_          63
    FileCount       14      46      71
    LAST_CHAR       27      40
    NULL            68
    Success_        30      47
    _A_ANY          30
    _A_SUBDIR       34
    argc            60      65      67
    argv            60      66      68
    do_dir          21      43      69
    ff              24      30      32      34      42      47
    ff_attr         34
    ff_name         32      42
    h        9      10
    main            60
    path            21      26      39
    printf          70
    recurse         63      68      69
    recurse_flag            21      37      43
    search          23      26      27      28      29      30
    stdio            9
    strcat          28      29      41      42
    strchr          68
    strcpy          26      39
    string          10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** FILECAT.C - Adds one file onto another vertically, as with a column
** block in QEdit.
**
** Must be compiled in Compact or Large memory models to use larger
** files.
**
** Public Domain by Chad Wallace, 1996
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "filecat.h"

#ifndef FALSE
#define FALSE 0
#define TRUE 1
#endif

/*
** filecat() error codes
*/

enum {
      FC_SUCCESS = 0,         /* Success */
      FC_RDEST,               /* Error reading dest file */
      FC_WDEST,               /* Error writing dest file */
      FC_SRC,                 /* Error with src file */
      FC_MEM,                 /* Out of memory */
      FC_LINES,               /* Too many lines (increase max_lines) */
      FC_LINE                 /* Line too long (increase line_max) */
};

/*
** These are global to save overhead when calling cleanup function
*/

static char ** str_arr;
static char * temp_str;

/*
** Frees all allocated memory and closes specified file
*/

static void cleanup(int tot_lines, FILE * fp)
{
      if (str_arr)
      {
            for ( ; tot_lines >= 0; tot_lines--)
                  free(str_arr[tot_lines]);
            free(str_arr);
      }
      if (temp_str)
            free(temp_str);
      if (fp)
            fclose(fp);
}

int filecat(char * dest_file, char * src_file, int line_max, int max_lines)
{
      int i, tot_lines;
      unsigned int lines_len;
      FILE * fp;

      if ((temp_str = malloc(line_max + 1)) == NULL)
            return FC_MEM;

      /* Allocate memory for pointers to line strings */

      if ((str_arr = calloc(max_lines + 1, sizeof(char *))) == NULL)
      {
            free(temp_str);
            return FC_MEM;
      }

      /* Open destination file */

      if ((fp = fopen(dest_file, "rt")) == NULL)
      {
            free(temp_str);
            free(str_arr);
            return FC_RDEST;
      }

      /* Read destination file into string array line-by-line */

      for (i = 0;
           (fgets(temp_str, line_max + 1, fp) != NULL) && (i <= max_lines);
           i++)
      {
            /* Strip trailing newline from line read */

            if (temp_str[strlen(temp_str) - 1] == '\n')
                  temp_str[strlen(temp_str) - 1] = '\0';

            /* Allocate memory */

```

```
101 |             if ((str_arr[i] = malloc(line_max + 1)) == NULL)
102 |             {
103 |                   /* Clean up and return memory error */
104 |
105 |                   cleanup(i, fp);
106 |
107 |                   return FC_MEM;
108 |             }
109 |
110 |             /* Copy the string to its new home */
111 |
112 |             strcpy(str_arr[i], temp_str);
113 |       }
114 |
115 |       fclose(fp);
116 |
117 |       if (i > max_lines)
118 |       {
119 |             /* Clean up and return too many lines error */
120 |
121 |             cleanup(i, NULL);
122 |
123 |             return FC_LINES;
124 |       }
125 |
126 |       /* Get length of longest line */
127 |
128 |       lines_len = max_line(str_arr);
129 |
130 |       /* Open source file */
131 |
132 |       if ((fp = fopen(src_file, "rt")) == NULL)
133 |       {
134 |             /* Clean up and return source file error */
135 |
136 |             cleanup(i, NULL);
137 |
138 |             return FC_SRC;
139 |       }
140 |
141 |       tot_lines = i;
142 |
143 |       /*
144 |       ** Get each line from src file and append to corresponding line from
145 |       ** dest file
146 |       */
147 |
148 |       for (i = 0;
149 |            (fgets(temp_str, line_max + 1, fp) != NULL) && (i < max_lines);
150 |            i++)
151 |       {
152 |             int j;
153 |
154 |             /* Has this line been allocated yet? */
155 |
156 |             if (str_arr[i] == NULL)
157 |             {
158 |                   /* Allocate memory */
159 |
160 |                   if ((str_arr[i] = malloc(line_max + 1)) == NULL)
161 |                   {
162 |                         /* Clean up and return memory error */
163 |
164 |                         cleanup(tot_lines, fp);
165 |
166 |                         return FC_MEM;
167 |                   }
168 |
169 |                   /* Initialize string */
170 |
171 |                   str_arr[i][0] = '\0';
172 |
173 |                   tot_lines++;
174 |             }
175 |
176 |             /* Pad with spaces to make this line as long as longest */
177 |
178 |             for (j = strlen(str_arr[i]); j < lines_len; j++)
179 |                   str_arr[i][j] = ' ';
180 |             str_arr[i][j] = '\0';
181 |
182 |             /* Check size of resulting line when strcat'd */
183 |
184 |             if (lines_len + strlen(temp_str) > line_max)
185 |             {
186 |                   cleanup(tot_lines, fp);
187 |
188 |                   return FC_LINE;
189 |             }
190 |
191 |             strcat(str_arr[i], temp_str);
192 |       }
193 |
194 |       fclose(fp);
195 |
196 |       if (i > max_lines)
197 |       {
198 |             /* Clean up and return too many lines error */
199 |
200 |             cleanup(i, NULL);
```

```
201 |
202 |             return FC_LINES;
203 |       }
204 |
205 |       /* Open dest file again, to write this time */
206 |
207 |       if ((fp = fopen(dest_file, "wt")) == NULL)
208 |       {
209 |             /* Clean up and return source file error */
210 |
211 |             cleanup(i, NULL);
212 |
213 |             return FC_WDEST;
214 |       }
215 |
216 |       for (i = 0; i < tot_lines; i++)
217 |       {
218 |             if ((fputs(str_arr[i], fp) == EOF) /*|| (fputc('\n', fp) == EOF)*/)
219 |             {
220 |                   cleanup(i, fp);
221 |
222 |                   return FC_WDEST;
223 |             }
224 |       }
225 |
226 |       cleanup(i, fp);
227 |
228 |       return FC_SUCCESS;
229 | }
230 |
231 | int main(int argc, char ** argv)
232 | {
233 |       printf("Filecat returned %d\n", filecat(argv[1], argv[2], 2048, 4096));
234 |
235 |       return 0;
236 | }
```

```
TEXT STATISTICS

  5480 characters
   236 lines

LEXICAL STATISTICS

    34 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     5 constants [character]
     5 constants [string]
    21 constants [numeric]

SYMBOL TABLE

    EOF             218
    FALSE            18      19
    FC_LINE          34     188
    FC_LINES         33     123     202
    FC_MEM           32      69      76     107     166
    FC_RDEST         29      85
    FC_SRC           31     138
    FC_SUCCESS               28     228
    FC_WDEST         30     213     222
    FILE             48      66
    NULL             68      73      81      91     101     121     132     136     149     156     160
           200      207     211
    TRUE             20
    argc            231
    argv            231     233+
    calloc           73
    cleanup          48     105     121     136     164     186     200     211     220     226
    dest_file        62      81     207
    fclose           59     115     194
    fgets            91     149
    filecat          62     233
    fopen            81     132     207
    fp               48      58      59      66      81      91     105     115     132     149     164
           186      194     207     218     220     226
    fputs           218
    free             53      54      57      75      83      84
    h        13      14      15
    i        64      90      91      92     101     105     112     117     121     136     141     148
           149     150     156     160     171     178     179     180     191     196     200     211
           216+    218     220     226
    j       152     178+    179     180
    line_max         62      68      91     101     149     160     184
    lines_len        65     128     178     184
    main            231
    malloc           68     101     160
    max_line        128
    max_lines        62      73      91     117     149     196
    printf          233
    src_file         62     132
    stdio            13
    stdlib           14
    str_arr          41      50      53      54      73      84     101     112     128     156     160
           171      178     179     180     191     218
    strcat          191
    strcpy          112
    string           15
    strlen           96      97     178     184
    temp_str         42      56      57      68      75      83      91      96+     97+     112     149
           184      191
    tot_lines        48      52+     53      64     141     164     173     186     216
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header for FILECAT.C and MAXLINES.C
   5 | */
   6 |
   7 | unsigned int max_line(char ** str_array);            /* MAXLINES.C     */
   8 | int filecat(char * dest_file,                        /* FILECAT.C      */
   9 |             char * src_file,
  10 |             int line_max,
  11 |             int max_lines);
```

TEXT STATISTICS

```
   342 characters
    11 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    dest_file        8
    filecat          8
    line_max        10
    max_line         7
    max_lines       11
    src_file         9
    str_array        7
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** FILELIST.C - Creates a link list of files and returns number of
  5 | **              files found.
  6 | **
  7 | ** public domain by Phi Nguyen
  8 | */
  9 |
 10 | #include "filelist.h"
 11 |
 12 | /*
 13 | ** Free_FileList - This function simply frees each linked list item.
 14 | */
 15 |
 16 | void Free_FileList(FILELIST *p)
 17 | {
 18 |       FILELIST *next = p->next;
 19 |
 20 |       while (NULL != next)
 21 |       {
 22 |             p->next = next->next;
 23 |             free(next);
 24 |             next = p->next;
 25 |       }
 26 | }
 27 |
 28 | /*
 29 | ** Get_FileList - This function creates a linked list of input source
 30 | **                 files.  Wildcard specifications are accommodated.
 31 | */
 32 |
 33 | int Get_FileList(FILELIST *list, char **argv, int argc, int attr)
 34 | {
 35 |       int          i = 0,
 36 |                    nCount = 0;
 37 |       DOSFileData  file;
 38 |       FILELIST    *base = list,
 39 |                    *node;
 40 |
 41 |       for ( ; i < argc; i++)
 42 |       {
 43 |             if (!FIND_FIRST(argv[i], attr, &file)) do
 44 |             {
 45 |                   if (NULL == (node =(FILELIST *)malloc(LIST_SIZE)))
 46 |                   {
 47 |                         Free_FileList(list);
 48 |                         list = NULL;
 49 |                         return 0;
 50 |                   }
 51 |                   base->file = file;
 52 |                   base->next = node;
 53 |                   node->next = NULL;
 54 |                   base = node;
 55 |                   nCount++;
 56 |             } while (!FIND_NEXT(&file));
 57 |       }
 58 |       return nCount;
 59 | }
 60 |
 61 | #ifdef TEST
 62 |
 63 | #include <stdio.h>
 64 | #define _A_FILE (_A_NORMAL|_A_RDONLY|_A_HIDDEN|_A_SYSTEM|_A_ARCH)
 65 |
 66 | main(int argc, char *argv[])
 67 | {
 68 |       FILELIST  list,
 69 |                *plist = &list;
 70 |       int       i,
 71 |                 nFiles;
 72 |
 73 |       if (argc < 2)
 74 |       {
 75 |             puts("Usage: FILELIST filespec ....");
 76 |             return EXIT_FAILURE;
 77 |       }
 78 |
 79 |       nFiles = Get_FileList(plist, argv + 1, argc - 1, _A_FILE);
 80 |       if (nFiles >= 0)
 81 |       {
 82 |             for (i = 0; i < nFiles; i++)
 83 |             {
 84 |                   printf("%-12s %8li  %2u-%02u-%02u  %2u:%02u\n",
 85 |                         ff_name(&plist->file), ff_size(&plist->file),
 86 |                         ff_mo(&plist->file), ff_day(&plist->file),
 87 |                         ff_yr(&plist->file) + 80, ff_hr(&plist->file),
 88 |                         ff_min(&plist->file));
 89 |                   plist = plist->next;
 90 |             }
 91 |             Free_FileList(&list);
 92 |       }
 93 |       else
 94 |       {
 95 |             puts("Get_FileList() failed");
 96 |             return EXIT_FAILURE;
 97 |       }
 98 |       return EXIT_SUCCESS;
 99 | }
100 |
```

```
 101 | #endif /* TEST */


TEXT STATISTICS

  2570 characters
   101 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     4 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    DOSFileData             37
    EXIT_FAILURE            76      96
    EXIT_SUCCESS            98
    FILELIST        16      18      33      38      45      68
    FIND_FIRST              43
    FIND_NEXT       56
    Free_FileList           16      47      91
    Get_FileList            33      79
    LIST_SIZE       45
    NULL            20      45      48      53
    TEST            61
    _A_FILE         64      79
    _A_NORMAL|_A_RDONLY|_A_HIDDEN|_A_SYSTEM|_A_ARCH         64
    argc            33      41      66      73      79
    argv            33      43      66      79
    attr            33      43
    base            38      51      52      54
    ff_day          86
    ff_hr           87
    ff_min          88
    ff_mo           86
    ff_name         85
    ff_size         85
    ff_yr           87
    file            37      43      51+     56      85+     86+     87+     88
    free            23
    h       63
    i       35      41+     43      70      82+
    list            33      38      47      48      68      69      91
    main            66
    malloc          45
    nCount          36      55      58
    nFiles          71      79      80      82
    next            18+     20      22+     23      24+     52      53      89
    node            39      45      52      53      54
    p       16      18      22      24
    plist           69      79      85+     86+     87+     88      89+
    printf          84
    puts            75      95
    stdio           63
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  FILELIST.H
   5 | */
   6 |
   7 | #ifndef FILELIST_H
   8 | #define FILELIST_H
   9 |
  10 | #include <stdlib.h>
  11 | #include <string.h>
  12 | #include "dirport.h"
  13 |
  14 | typedef struct _filelist {
  15 |       DOSFileData file;
  16 |       struct _filelist *next;
  17 | } FILELIST;
  18 |
  19 | #define LIST_SIZE sizeof(FILELIST)
  20 |
  21 | int Get_FileList(FILELIST *list, char **argv, int agrc, int attr);
  22 | void Free_FileList(FILELIST *list);
  23 |
  24 | #endif /* FILELIST_H */
```

TEXT STATISTICS

```
   447 characters
    24 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             15
    FILELIST        17      19      21      22
    FILELIST_H               7       8
    Free_FileList           22
    Get_FileList            21
    LIST_SIZE       19
    _filelist       14      16
    agrc             21
    argv             21
    attr             21
    file             15
    h       10       11
    list             21      22
    next             16
    stdlib           10
    string           11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** FILES.C:  A program to determine the number of file handles
**
** Released in to the Public Domain by Matthew Hunt @ 1:129/135, in the
** hopes that no "programmer" will be so lazy that he|she simple reads
** the CONFIG.SYS file ever again.
**
** Any improvements and modifications are welcome, but I ask that all
** modified versions also be placed into the Public Domain.
**
** Information on the List of Lists and SFT format was provided by
** PC Magazine November 26, 1991, and PC Interrupts by Ralf Brown
** and Jim Kyle.  FILES.C was originally written for Power C.
**
** Modifications for other compiler support by Bob Stout @ 1:106/2000.6
*/

#include <dos.h>
#include "dosfiles.h"
#include "mk_fp.h"

/*
** Walk the SFT linked list, counting file handles as we go
*/

int files(void)
{
      struct SFT_HEADER (FAR *sft);
      unsigned int segment, offset;
      int count=0;
      union REGS regs;
      struct SREGS sregs;

      /* Get ptr to List of Lists in ES:DX */

      regs.x.ax = 0x5200;
      segread(&sregs);
      intdosx(&regs, &regs, &sregs);

      /* Get ssss:oooo to first SFT  */

      segment = *((unsigned FAR *)(MK_FP(sregs.es, regs.x.bx + 6)));
      offset  = *((unsigned FAR *)(MK_FP(sregs.es, regs.x.bx + 4)));

      /* Point our structure to it.  */

      sft = MK_FP(segment, offset);

      do
      {
            count += sft->number;               /* Add the number of FILES */
            sft = sft->next;                    /* Point to next one       */
      } while(FP_OFF(sft->next) != 0x0FFFF);    /* Last one in the chain   */

      /* Add the FILES for the last entry */

      count += sft->number;
      return count;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Number of FILES entries: %d", files());
    return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1967 characters
    74 lines

LEXICAL STATISTICS

    11 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     3 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    EXIT_SUCCESS            71
    FAR             30      44      45
    FP_OFF          55
    MK_FP           44      45      49
    REGS            33
    SFT_HEADER              30
    SREGS           34
    TEST            63
    ax              38
    bx              44      45
    count           32      53      59      60
    dos             20
    es              44      45
    files           28      70
    h       20      65      66
    intdosx         40
    main            68
    next            54      55
    number          53      59
    offset          31      45      49
    printf          70
    regs            33      38      40+     44      45
    segment         31      44      49
    segread         39
    sft             30      49      53      54+     55      59
    sregs           34      39      40      44      45
    stdio           65
    stdlib          66
    x       38      44      45
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  FILNAMES.H - Header file for SNIPPETS DOS file name processing functions.
  5 | */
  6 |
  7 | #ifndef FILNAMES__H
  8 | #define FILNAMES__H
  9 |
 10 | #include "sniptype.h"
 11 |
 12 | char *fln_fix(char *path);
 13 | int   flnorm(char *in_name, char *out_name);
 14 | char *unix2dos(char *path);
 15 | char *dos2unix(char *path);
 16 | char *chgext(char *path, char *oldext, char *newext);
 17 |
 18 | Boolean_T valid_fname (const char *fname, Boolean_T wild_check);
 19 |
 20 | /*
 21 | **  FNSPLIT.C prototypes and definitions
 22 | */
 23 |
 24 | #define Extension_ 1
 25 | #define Filename_  2
 26 | #define Directory_ 4
 27 | #define Drive_     8
 28 | #define Wildname_  16
 29 | #define Wildpath_  32
 30 |
 31 | Boolean_T has_wild(char *pname);
 32 | int       fnSplit(char *spec, char *drive, char *pname, char *path,
 33 |                   char *fname, char *name, char *ext);
 34 | char     *fnMerge(char *spec, char *drive, char *pname, char *path,
 35 |                   char *fname, char *name, char *ext) ;
 36 |
 37 | /*
 38 | **  Macros for backwards compatibility with previous SNIPPETS, also more
 39 | **  compatible with Borland psplit(), Microsoft _splitpath(), et al.
 40 | */
 41 |
 42 | #define psplit(str,drv,dir,nam,ext) fnSplit(str,drv,NULL,dir,NULL,nam,ext)
 43 | #define pmerge(str,drv,dir,nam,ext) fnMerge(str,drv,NULL,dir,NULL,nam,ext)
 44 |
 45 | #endif /* FILNAMES__H */
```

TEXT STATISTICS

```
  1273 characters
    45 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
     1 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       18+     31
    Directory_              26
    Drive_          27
    Extension_              24
    FILNAMES__H              7       8
    Filename_       25
    NULL            42+     43+
    Wildname_       28
    Wildpath_       29
    chgext          16
    dir             42+     43+
    dos2unix        15
    drive           32      34
    drv             42+     43+
    ext             33      35      42+     43+
    fln_fix         12
    flnorm          13
    fnMerge         34      43
    fnSplit         32      42
    fname           18      33      35
    has_wild        31
    in_name         13
    nam             42+     43+
    name            33      35
    newext          16
    oldext          16
    out_name        13
    path            12      14      15      16      32      34
    pmerge          43
    pname           31      32      34
    psplit          42
    spec            32      34
    str             42+     43+
    unix2dos        14
    valid_fname             18
    wild_check              18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FLNORM.C - Normalize DOS file names
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdio.h>
#include <string.h>
#ifdef __TURBOC__
 #include <dir.h>
#else
 #include <direct.h>
#endif
#include <dos.h>
#include <io.h>
#include "sniptype.h"
#include "filnames.h"

#if defined(_WIN32) || defined(WIN32) || defined(__NT__)
 #define chdrv _chdrive
#else
 #include "dosfiles.h"
#endif

int flnorm(char *in_name, char *out_name)
{
      Boolean_T dir_flag = False_, new_drv = False_;
      int status = 0, level = 0;
      char *p, *out;
      static char drive[2][3];
      static char file[14];
      static char I_am_here[FILENAME_MAX];
      static char I_am_there[FILENAME_MAX];
      static char remember[FILENAME_MAX];

      getcwd(I_am_here, FILENAME_MAX);
      if (!in_name || !in_name[0])
      {
            strcpy(out_name, I_am_here);
            goto ERRexit;
      }
      strncpy(drive[0], I_am_here, 2);
      drive[0][2] = '\0';
      if (':' == in_name[1])
      {     /* If a drive is specified                          */
            if (chdrv(in_name[0]))
            {     /* If the drive is invalid                    */
                  status = Error_;
                  goto ERRexit;
            }
            new_drv = True_;
            getcwd(remember, FILENAME_MAX);
            strncpy(drive[1], remember, 2);
            drive[1][2] = '\0';
      }
      else
      {     /* If a drive isn't specified                       */
            if (NULL != (p = strchr(in_name, ':')))
            {     /* If filename is illegal                     */
                  status = Error_;
                  goto ERRexit;
            }
      }
      unix2dos(in_name);
      if (new_drv)
      {
            if ('\\' == in_name[2])
                  strcpy(out_name, drive[1]);
            else
            {
                  strcpy(out_name, remember);
                  if ('\\' != LAST_CHAR(remember))
                        strcat(out_name, "\\");
            }
      }
      else
      {
            strcpy(out_name, drive[0]);
            if ('\\' != *in_name)
            {
                  strcat(out_name, I_am_here);
                  if ('\\' != LAST_CHAR(I_am_here))
                        strcat(out_name, "\\");
            }
      }
      strcat(out_name, &in_name[(new_drv) ? 2 : 0]);
      fln_fix(out_name);
      out = &out_name[2];
      if (!(*out))
            goto ERRexit;
      while ('\\' == LAST_CHAR(out))
      {     /* Strip trailing `\'s                              */
            LAST_CHAR(out_name) = '\0';
```

```
101 |             dir_flag = True_;
102 |       }
103 |       if (!(*out))
104 |       {
105 |             if (dir_flag)
106 |             {
107 |                   strcat(out, "\\");
108 |                   goto ERRexit;
109 |             }
110 |             else  goto BADPATH;
111 |       }
112 |       if (NULL != (p = strrchr(out_name, '\\')))
113 |             strcpy(file, p);        /* Save filename              */
114 |       if (chdir(out))
115 |       {     /* If can't move to path                             */
116 |             if ((!dir_flag) && p)
117 |             {     /* If there was a separate path                */
118 |                   *p = '\0';
119 |                   if (!(*out))
120 |                   {     /* Back at the root, handle it          */
121 |                         strcpy(p, "\\");
122 |                         strcpy(file, &file[1]);
123 |                   }
124 |                   if (chdir(out))
125 |                   {     /* If we can't move to path             */
126 |                         *p = '\\';
127 |                         goto BADPATH;
128 |                   }
129 |                   ++level;          /* Flag we stripped name     */
130 |             }
131 |             else
132 |             {     /* No path as specified                        */
133 |                   if (p)
134 |                   {
135 | BADPATH:                status = Error_;
136 |                         goto ERRexit;
137 |                   }
138 |             }
139 |       }
140 |       getcwd(I_am_there, FILENAME_MAX); /* Get normalized path       */
141 |       strupr(I_am_there);
142 |       strcpy(out_name, I_am_there);
143 |       if (level)
144 |             strcat(out_name, file);
145 | ERRexit:
146 |       if (new_drv)
147 |       {
148 |             chdir(remember);
149 |             chdrv(I_am_here[0]);
150 |       }
151 |       chdir(I_am_here);
152 |       if (status)
153 |             out_name[0] = '\0';
154 |       return status;
155 | }
```

TEXT STATISTICS

```
  4744 characters
   155 lines
```

LEXICAL STATISTICS

```
    15 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
    14 constants [character]
     7 constants [string]
    25 constants [numeric]
```

SYMBOL TABLE

```
    BADPATH        110     127     135
    Boolean_T       34
    ERRexit         47      56      68      97     108     136     145
    Error_          55      67     135
    FILENAME_MAX            39      40      41      43      59     140
    False_          34+
    I_am_here       39      43      46      49      88      89     149     151
    I_am_there              40     140     141     142
    LAST_CHAR       79      89      98     100
    NULL            65     112
    True_           58     101
    WIN32           26
    _WIN32          26
    __NT__          26
    __TURBOC__              16
    _chdrive        27
    chdir          114     124     148     151
    chdrv           27      53     149
    defined         26+
    dir             17
    dir_flag        34     101     105     116
    direct          19
    dos             21
    drive           37      49      50      60      61      75      85
    file            38     113     122+    144
    fln_fix         94
    flnorm          32
    getcwd          43      59     140
    h       14      15      17      19      21      22
    in_name         32      44+     51      53      65      71      74      86      93
    io              22
    level           35     129     143
    new_drv         34      58      72      93     146
    out             36      95      96      98     103     107     114     119     124
    out_name        32      46      75      78      80      85      88      90      93      94      95
           100     112     142     144     153
    p       36      65     112     113     116     118     121     126     133
    remember        41      59      60      78      79     148
    status          35      55      67     135     152     154
    stdio           14
    strcat          80      88      90      93     107     144
    strchr          65
    strcpy          46      75      78      85     113     121     122     142
    string          15
    strncpy         49      60
    strrchr        112
    strupr         141
    unix2dos        71
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FLN_FIX.C
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <io.h>
#include "sniptype.h"
#include "filnames.h"

/*************************************************************/
/*                                                           */
/*  Function to `crunch' dot directories and check for       */
/*  DOS-valid path strings. Drive specifiers in the path     */
/*  ignored.                                                 */
/*                                                           */
/*************************************************************/

char *fln_fix(char *path)
{
      Boolean_T dir_flag = False_, root_flag = False_;
      char *r, *p, *q, *s;

      if (path)
            strupr(path);

      /* Ignore leading drive specs   */

      if (NULL == (r = strrchr(path, ':')))
            r = path;
      else  ++r;

      unix2dos(r);                     /* Convert Unix to DOS style    */

      while ('\\' == *r)               /* Ignore leading backslashes   */
      {
            if ('\\' == r[1])
                  strcpy(r, &r[1]);
            else
            {
                  root_flag = True_;
                  ++r;
            }
      }

      p = r;                           /* Change "\\" to "\"           */
      while (NULL != (p = strchr(p, '\\')))
      {
            if ('\\' ==  p[1])
                  strcpy(p, &p[1]);
            else  ++p;
      }

      while ('.' == *r)                /* Scrunch leading ".\"         */
      {
            if ('.' == r[1])
            {
                  /* Ignore leading ".."                           */

                  for (p = (r += 2); *p && (*p != '\\'); ++p)
                        ;
            }
            else
            {
                  for (p = r + 1 ;*p && (*p != '\\'); ++p)
                        ;
            }
            strcpy(r, p + ((*p) ? 1 : 0));
      }

      while ('\\' == LAST_CHAR(path))   /* Strip trailing backslash     */
      {
            dir_flag = True_;
            LAST_CHAR(path) = '\0';
      }

      s = r;

      /* Look for "\." in path        */

      while (NULL != (p = strstr(s, "\\.")))
      {
            if ('.' == p[2])
            {
                  /* Execute this section if ".." found               */

                  q = p - 1;
                  while (q > r)           /* Backup one level          */
                  {
                        if (*q == '\\')
```

```
101 |                                 break;
102 |                         --q;
103 |                   }
104 |                   if (q > r)
105 |                   {
106 |                         strcpy(q, p + 3);
107 |                         s = q;
108 |                   }
109 |                   else if ('.' != *q)
110 |                   {
111 |                         strcpy(q + ((*q == '\\') ? 1 : 0),
112 |                               p + 3 + ((*(p + 3)) ? 1 : 0));
113 |                         s = q;
114 |                   }
115 |                   else  s = ++p;
116 |
117 |             }
118 |             else
119 |             {
120 |                   /* Execute this section if "." found                  */
121 |
122 |                   q = p + 2;
123 |                   for ( ;*q && (*q != '\\'); ++q)
124 |                         ;
125 |                   strcpy (p, q);
126 |             }
127 |       }
128 |
129 |       if (root_flag)  /* Embedded ".." could have bubbled up to root  */
130 |       {
131 |             for (p = r; *p && ('.' == *p || '\\' == *p); ++p)
132 |                   ;
133 |             if (r != p)
134 |                   strcpy(r, p);
135 |       }
136 |
137 |       if (dir_flag)
138 |             strcat(path, "\\");
139 |       return path;
140 | }
```

```
TEXT STATISTICS

  4045 characters
   140 lines

LEXICAL STATISTICS

    21 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
    18 constants [character]
     4 constants [string]
    19 constants [numeric]

SYMBOL TABLE

    Boolean_T       31
    False_          31+
    LAST_CHAR       81      84
    NULL            39      57      91
    True_           51      83
    dir_flag        31      83     137
    dos             16
    fln_fix         29
    h       14      15      16      17
    io              17
    p       32      56      57+     59      60+     61      70+     75+     78+     91      93      97
           106     112+    115     122     125     131+    133     134
    path            29      34      35      39      40      81      84     138     139
    q       32      97      98     100     102     104     106     107     109     111+    113     122
           123+    125
    r       32      39      40      41      43      45      47      48+     52      56      64      66
            70      75      78      87      98     104     131     133     134
    root_flag       31      51     129
    s       32      87      91     107     113     115
    stdio           14
    strcat         138
    strchr          57
    strcpy          48      60      78     106     111     125     134
    string          15
    strrchr         39
    strstr          91
    strupr          35
    unix2dos        43
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FLOPCOPY.C
**
**  Copy a floppy to a hard disk directory with directory recursion
**  Public domain, uses functions from SNIPPETS.
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sniptype.h"
#include "dirport.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif
#include "snipfile.h"

void do_dir(char *, char *);

/*
**  Copy a floppy to an HD subdirectory
*/

int main(int argc, char *argv[])
{
      char fdrv[4] = "A:\\", target[FILENAME_MAX];

      if (3 > argc)
      {
            puts("Usage: FLOPCOPY drive_letter subdir");
            puts("where: drive_letter is \"A\" or \"B\" (colon optional)");
            puts("       subdir is drive:dir target, e.g. \"C:\\FLOPSTUF\"");
            return EXIT_FAILURE;
      }
      *fdrv = *argv[1];
      strcpy(target, argv[2]);
      if ('\\' != LAST_CHAR(target))
            strcat(target, "\\");

      do_dir(fdrv, target);
      return EXIT_SUCCESS;
}

/*
**  Process a directory (SNIPPETS: Treedir.C, modified)
*/

void do_dir(char *from, char *to)
{
      char search[FILENAME_MAX], new[FILENAME_MAX], newto[FILENAME_MAX];
      DOSFileData ff;

      strcat(strcpy(search, from), "*.*");
      if (Success_ == FIND_FIRST(to, _A_ANY, &ff))
      {
            if (0 == (ff_attr(&ff) & _A_SUBDIR))
            {
                  printf("*** %s Exists and is not a directory!\n", to);
                  return;
            }
      }
      else
      {
            strcpy(newto, to);
            if ('\\' == LAST_CHAR(newto))
                  LAST_CHAR(newto) = NUL;
            mkdir(newto);
      }
      if (Success_ == FIND_FIRST(search, _A_ANY, &ff)) do
      {
            if (ff_attr(&ff) & _A_SUBDIR && '.' != *ff_name(&ff))
            {
                  strcat(strcat(strcpy(new, from), ff_name(&ff)), "\\");
                  strcat(strcat(strcpy(newto, to), ff_name(&ff)), "\\");
                  do_dir(new, newto);
            }
            else
            {
                  char file1[FILENAME_MAX], file2[FILENAME_MAX];

                  if ((ff_attr(&ff) & (_A_SUBDIR | _A_VOLID)) ||
                        '.' == *ff_name(&ff))
                  {
                        continue;
                  }
                  strcat(strcpy(file1, from), ff_name(&ff));
                  strcat(strcpy(file2, to), ff_name(&ff));
                  if (Success_ != file_copy(file1, file2))
                        printf("*** Unable to copy %s to %s\n", file1, file2);
                  else  printf("Copied %s to %s\n", file1, file2);
            }
      } while (Success_ == FIND_NEXT(&ff));
}
```

TEXT STATISTICS

```
  2797 characters
    97 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     4 constants [character]
    15 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             55
    EXIT_FAILURE            37
    EXIT_SUCCESS            45
    FILENAME_MAX            30      54+     83+
    FIND_FIRST              58      73
    FIND_NEXT       96
    LAST_CHAR       41      69      70
    MSDOS           15
    NUL             70
    Success_        58      73      92      96
    _A_ANY          58      73
    _A_SUBDIR       60      75      85
    _A_VOLID        85
    __MSDOS__       15
    argc            28      32
    argv            28      39      40
    defined         15+
    do_dir          22      44      52      79
    fdrv            30      39      44
    ff              55      58      60      73      75+     77      78      85      86      90      91
            96
    ff_attr         60      75      85
    ff_name         75      77      78      86      90      91
    file1           83      90      92      93      94
    file2           83      91      92      93      94
    file_copy       92
    from            52      57      77      90
    h       10      11      12      18
    main            28
    mkdir           71
    newto           54      68      69      70      71      78      79
    printf          62      93      94
    puts            34      35      36
    search          54      57      73
    stdio           10
    stdlib          12
    strcat          42      57      77+     78+     90      91
    strcpy          40      57      68      77      78      90      91
    string          11
    target          30      40      41      42      44
    to              52      58      62      68      78      91
    unistd          18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FMEMMEM.C - A strstr() work-alike for large non-text buffers
**
**  public domain by Bob Stout
*/

#include "snip_str.h"
#include "snpdosys.h"                    /* For addptr() & farnormal() */

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

void FAR *fmemmem(const void FAR *buf,
                  const void FAR *pattern,
                  long            buflen,
                  long            len)
{
      long i, j;
      char FAR *bf = (char FAR *)buf, FAR *pt = (char FAR *)pattern;

      if (len > buflen)
            return (void FAR *)NULL;

      for (i = 0L; i <= (buflen - len); ++i)
      {
            for (j = 0L; j < len; ++j)
            {
                  /*
                  ** Not all compilers support huge pointers the same way
                  ** (or at all!), so add the offsets to the pointers,
                  ** normalize them, then dereference them.
                  */

                  char FAR *pp = addptr(pt, j);       /* &pt[j]     */
                  char FAR *bb = addptr(bf, i + j);   /* &bf[i + j]     */

                  if (*pp != *bb)                     /* pt[j]==bf[i+j] */
                        break;
            }
            if (j == len)
                  return addptr(bf, i);
      }
      return (void FAR *)NULL;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

main()
{
      char FAR buf[13] = "\0""12344567890\x1b";
      char FAR a[3] = "456";
      char FAR b[3] = "\0""12";
      char FAR c[3] = "90\x1b";
      char FAR d[3] = "ABC";
      char FAR e[3] = "0\x1b""\0";
      char FAR f[1] = "\x1b";
      char FAR *ptr;
      long lp, lb;

      if (NULL == (ptr = fmemmem(buf, a, 13L, 3L)))
            puts("a not found in buf");
      else
      {
            lp = (long)farnormal(ptr);
            lb = (long)farnormal(buf);
            printf("a found in buf at posn %ld\n", lp - lb);
      }

      if (NULL == (ptr = fmemmem(buf, b, 13L, 3L)))
            puts("b not found in buf");
      else
      {
            lp = (long)farnormal(ptr);
            lb = (long)farnormal(buf);
            printf("b found in buf at posn %ld\n", lp - lb);
      }

      if (NULL == (ptr = fmemmem(buf, c, 13L, 3L)))
            puts("c not found in buf");
      else
      {
            lp = (long)farnormal(ptr);
            lb = (long)farnormal(buf);
            printf("c found in buf at posn %ld\n", lp - lb);
      }

      if (NULL == (ptr = fmemmem(buf, d, 13L, 3L)))
            puts("d not found in buf");
      else
      {
            lp = (long)farnormal(ptr);
```

```
101 |              lb = (long)farnormal(buf);
102 |              printf("d found in buf at posn %ld\n", lp - lb);
103 |       }
104 |
105 |       if (NULL == (ptr = fmemmem(buf, e, 13L, 3L)))
106 |              puts("e not found in buf");
107 |       else
108 |       {
109 |              lp = (long)farnormal(ptr);
110 |              lb = (long)farnormal(buf);
111 |              printf("e found in buf at posn %ld\n", lp - lb);
112 |       }
113 |
114 |       if (NULL == (ptr = fmemmem(buf, f, 13L, 1L)))
115 |              puts("f not found in buf");
116 |       else
117 |       {
118 |              lp = (long)farnormal(ptr);
119 |              lb = (long)farnormal(buf);
120 |              printf("f found in buf at posn %ld\n", lp - lb);
121 |       }
122 |
123 |       return 0;
124 | }
125 |
126 | #endif /* TEST */
```

TEXT STATISTICS

```
  3440 characters
   126 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
    25 constants [string]
    22 constants [numeric]
```

SYMBOL TABLE

```
    FAR             16+     17      22+     25      37      38      46      59      60      61      62
           63       64      65      66
    NULL            25      46      69      78      87      96     105     114
    TEST            53
    __cplusplus             12+     49+
    a       60      69
    addptr          37      38      44
    b       61      78
    bb              38      40
    bf              22      38      44
    buf             16      22      59      69      74      78      83      87      92      96     101
           105     110     114     119
    buflen          18      24      27
    c       62      87
    d       63      96
    defined         12      49
    e       64     105
    f       65     114
    farnormal       73      74      82      83      91      92     100     101     109     110     118
           119
    fmemmem         16      69      78      87      96     105     114
    h       55
    i       21      27+     38      44
    j       21      29+     37      38      43
    lb              67      74      75      83      84      92      93     101     102     110     111
           119     120
    len             19      24      27      29      43
    lp              67      73      75      82      84      91      93     100     102     109     111
           118     120
    main            57
    pattern         17      22
    pp              37      40
    printf          75      84      93     102     111     120
    pt              22      37
    ptr             66      69      73      78      82      87      91      96     100     105     109
           114     118
    puts            70      79      88      97     106     115
    stdio           55
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FMEMOPS.C - Emulate MSC's far memory functions in ZTC++ & early BC++
**
**  Original Copyright 1988-1992 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#if defined(__ZTC__) && !defined(__SC__)

#include <stdlib.h>
#include <dos.h>
#include "fmemops.h"

/*
**  Don't #include <string.h> to avoid incompatible prototypes
*/

#if __cplusplus
 extern "C" {
#endif

void CDECL movedata(unsigned,unsigned,unsigned,unsigned,size_t);

#if __cplusplus
 }
#endif

typedef unsigned char FAR *FarBytePtr;

void FAR * _fmemcpy(void FAR *dest, void FAR *src, size_t count)
{
      movedata(FP_SEG(src), FP_OFF(src), FP_SEG(dest), FP_OFF(dest), count);
      return dest;
}

void FAR * _fmemmove(void FAR *dest, void FAR *src, size_t count)
{
      void FAR *target =  dest;
      FarBytePtr to = (FarBytePtr)dest, from = (FarBytePtr)src;

      if (src >= dest)
            _fmemcpy(dest, src, count);
      else  for (to += count, from += count; count; --count)
                  *--to = *--from;
      return target;
}

void FAR * _fmemset(void FAR *dest, int ch, size_t count)
{
      void FAR *target =  dest;
      FarBytePtr to = (FarBytePtr)dest;

      for ( ; count; --count)
            *to++ = (unsigned char) ch;
      return target;
}

#endif /* __ZTC__ */
```

TEXT STATISTICS

```
  1590 characters
    64 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CDECL           28
    FAR             34      36+     42+     44      54+     56
    FP_OFF          38+
    FP_SEG          38+
    FarBytePtr              34      45+     57+
    __SC__          14
    __ZTC__         14
    __cplusplus             24      30
    _fmemcpy        36      48
    _fmemmove       42
    _fmemset        54
    ch              54      60
    count           36      38      42      48      49+     54      59+
    defined         14+
    dest            36      38+     39      42      44      45      47      48      54      56      57
    dos             17
    from            45      49      50
    h       16      17
    movedata        28      38
    size_t          28      36      42      54
    src             36      38+     42      45      47      48
    stdlib          16
    target          44      51      56      61
    to              45      49      50      57      60
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  FMEMOPS.H - SNIPPETS header file for FMEMOPS.C
   5 | */
   6 |
   7 | #ifndef FMEMOPS__H
   8 | #define FMEMOPS__H
   9 |
  10 | #include <stddef.h>
  11 | #include "extkword.h"
  12 |
  13 | typedef unsigned char FAR *FarBytePtr;
  14 |
  15 | void FAR * _fmemcpy(void FAR *dest, void FAR *src, size_t count);
  16 | void FAR * _fmemmove(void FAR *dest, void FAR *src, size_t count);
  17 | void FAR * _fmemset(void FAR *dest, int ch, size_t count);
  18 |
  19 | #endif /* FMEMOPS__H */


TEXT STATISTICS

   458 characters
    19 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    FAR             13      15+     16+     17+
    FMEMOPS__H               7       8
    FarBytePtr              13
    _fmemcpy        15
    _fmemmove       16
    _fmemset        17
    ch              17
    count           15      16      17
    dest            15      16      17
    h       10
    size_t          15      16      17
    src             15      16
    stddef          10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  FMTMONEY.C - Format a U.S. dollar value into a numeric string
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include <stdlib.h>
 11 | #include <string.h>
 12 | #include <math.h>
 13 | #include "snipmath.h"
 14 |
 15 | #define Form(s,a) bufptr += sprintf(bufptr, s, a)
 16 |
 17 | static char buf[256], *bufptr;
 18 |
 19 | static char *units[] = {"Zero", "One", "Two", "Three", "Four",
 20 |                         "Five", "Six", "Seven", "Eight", "Nine",
 21 |                         "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen",
 22 |                         "Fifteen", "Sixteen", "Seventeen", "Eighteen",
 23 |                         "Nineteen"},
 24 |
 25 |             *tens[]  = {"Twenty", "Thirty", "Forty", "Fifty", "Sixty",
 26 |                         "Seventy", "Eighty", "Ninety"};
 27 |
 28 | static void form_group(int, char *);
 29 |
 30 | /*
 31 | **  Call with double amount
 32 | **  Rounds cents
 33 | **  Returns string in a static buffer
 34 | */
 35 |
 36 | char *fmt_money(double amt)
 37 | {
 38 |       int temp;
 39 |       double dummy, cents = modf(amt, &dummy);
 40 |
 41 |       *buf = '\0';
 42 |       bufptr = buf;
 43 |
 44 |       temp = (int)(amt/1E12);
 45 |       if (temp)
 46 |       {
 47 |             form_group(temp, "Trillion");
 48 |             amt = fmod(amt, 1E12);
 49 |       }
 50 |
 51 |       temp = (int)(amt/1E9);
 52 |       if (temp)
 53 |       {
 54 |             form_group(temp, "Billion");
 55 |             amt = fmod(amt, 1E9);
 56 |       }
 57 |
 58 |       temp = (int)(amt/1E6);
 59 |       if (temp)
 60 |       {
 61 |             form_group(temp, "Million");
 62 |             amt = fmod(amt, 1E6);
 63 |       }
 64 |
 65 |       temp = (int)(amt/1E3);
 66 |       if (temp)
 67 |       {
 68 |             form_group(temp, "Thousand");
 69 |             amt = fmod(amt, 1E3);
 70 |       }
 71 |       form_group((int)amt, "");
 72 |
 73 |       if (buf == bufptr)
 74 |             Form("%s ", units[0]);
 75 |
 76 |       temp = (int)(cents * 100. + .5);
 77 |       sprintf(bufptr, "& %02d/100", temp);
 78 |
 79 |       return buf;
 80 | }
 81 |
 82 | /*
 83 | **  Process each thousands group
 84 | */
 85 |
 86 | static void form_group(int amt, char *scale)
 87 | {
 88 |       if (buf != bufptr)
 89 |             *bufptr++ = ' ';
 90 |
 91 |       if (100 <= amt)
 92 |       {
 93 |             Form("%s Hundred ", units[amt/100]);
 94 |             amt %= 100;
 95 |       }
 96 |       if (20 <= amt)
 97 |       {
 98 |             Form("%s", tens[(amt - 20)/10]);
 99 |             if (0 != (amt %= 10))
100 |             {
```

```
101 |                   Form("-%s ", units[amt]);
102 |               }
103 |                   else    Form("%s", " ");
104 |       }
105 |       else if (amt)
106 |       {
107 |             Form("%s ", units[amt]);
108 |       }
109 |
110 |       Form("%s", scale);
111 | }
112 |
113 | #ifdef TEST
114 |
115 | main(int argc, char *argv[])
116 | {
117 |       while (--argc)
118 |       {
119 |             double amt = atof(*(++argv));
120 |             printf("fmt_money(%g) = %s\n", amt, fmt_money(amt));
121 |       }
122 |       return EXIT_SUCCESS;
123 | }
124 |
125 | #endif /* TEST */
```

TEXT STATISTICS

```
  2733 characters
   125 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     2 constants [character]
    44 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS           122
    Form            15      74      93      98     101     103     107     110
    TEST           113
    a       15+
    amt             36      39      44      48+     51      55+     58      62+     65      69+     71
            86      91      93      94      96      98      99     101     105     107     119     120+
    argc           115     117
    argv           115     119
    atof           119
    buf             17      41      42      73      79      88
    bufptr          15+     17      42      73      77      88      89
    cents           39      76
    dummy           39+
    fmod            48      55      62      69
    fmt_money       36     120
    form_group              28      47      54      61      68      71      86
    h        9      10      11      12
    main           115
    math            12
    modf            39
    printf         120
    s       15+
    scale           86     110
    sprintf         15      77
    stdio            9
    stdlib          10
    string          11
    temp            38      44      45      47      51      52      54      58      59      61      65
            66      68      76      77
    tens            25      98
    units           19      74      93     101     107
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Originally published as part of the MicroFirm Function Library
**
**  Copyright 1990, Robert B.Stout
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Function to locate an unused user interrupt vector.
*/

#include "extkword.h"
#include "snpdosys.h"

#ifdef __ZTC__
 #include <int.h>
#else
 #include <dos.h>
 #ifdef __TURBOC__
  #define GETVECT getvect
 #else /* assume MSC */
  #define GETVECT _dos_getvect
 #endif
 #define FNULL (void (FAR *)())(0L)
#endif

unsigned findIslot(void)
{
#ifdef __ZTC__
      unsigned int_no, seg, ofs;

      for (int_no = 0x60; int_no < 0x6f; ++int_no)
      {
            int_getvector(int_no, &seg, &ofs);
            if (0U == (seg | ofs))
                  return int_no;
      }
#else /* MSC/BC/TC */
      unsigned int_no;

      for (int_no = 0x60; int_no < 0x6f; ++int_no)
      {
            if (FNULL != (void (FAR *)())GETVECT(int_no))
                  return int_no;
      }
#endif
      return 0;
}
```

TEXT STATISTICS

```
  1204 characters
    51 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
     2 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    FAR             27      46
    FNULL           27      46
    GETVECT         23      25      46
    __TURBOC__              22
    __ZTC__         18      32
    _dos_getvect            25
    dos             21
    findIslot       30
    getvect         23
    h       19      21
    int_getvector           37
    int_no          33      35+     37      39      42      44+     46      47
    ofs             33      37      38
    seg             33      37      38
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************/
/*                                                                           */
/*  FNSPLIT.C                                                                */
/*                                                                           */
/*  Contains routines for parsing file/path names.                           */
/*                                                                           */
/*  Original Copyright 1989-96 by Robert B. Stout as part of                 */
/*  the MicroFirm Function Library (MFL)                                     */
/*                                                                           */
/*  The user is granted a free limited license to use this source file       */
/*  to create royalty-free programs, subject to the terms of the             */
/*  license restrictions specified in the LICENSE.MFL file.                  */
/*                                                                           */
/*****************************************************************************/

#include <string.h>
#include <stdlib.h>
#include "filnames.h"

/*****************************************************************/
/*                                                               */
/*  has_wild()                                                   */
/*                                                               */
/*  Checks a string for wildcard characters ('?' and '*')        */
/*                                                               */
/*  Arguments 1 - String to check                                */
/*                                                               */
/*  Returns: True_ if string contains wildcards, else False_     */
/*                                                               */
/*  Side Effects: None                                           */
/*                                                               */
/*****************************************************************/

Boolean_T has_wild(char *pname)
{
      if (NULL != strchr(pname, '*') || NULL != strchr(pname, '?'))
            return True_;
      else  return False_;
}

/*****************************************************************/
/*                                                               */
/*  fnSplit()                                                    */
/*                                                               */
/*  Splits file specifications into component parts. Similar to  */
/*  compiler-specific fnsplit() or _splitpath().                 */
/*                                                               */
/*  Arguments 1 - Original file specification                    */
/*            2 - Buffer to receive drive spec                   */
/*            3 - Buffer to receive drive/path spec              */
/*            4 - Buffer to receive path spec                    */
/*            5 - Buffer to receive name.ext spec                */
/*            6 - Buffer to receive name spec                    */
/*            7 - Buffer to receive ext spec                     */
/*                                                               */
/*  Returns: Bit map as follows (see defines in RBS.H):          */
/*           Extension_ - File spec included extension           */
/*           Filename_  - File spec did not end in '\'           */
/*           Directory_ - File spec included a path              */
/*           Drive_     - File spec included a drive spec        */
/*           Wildname_  - File name included wildcards (*.?)     */
/*           Wildpath_  - File path included wildcards (*.?)     */
/*                                                               */
/*  Side Effects: Calls unix2dos() to convert '/' to '\'         */
/*                                                               */
/*  Notes: Passing NULL in arguments 2-7 causes fnsplit() to     */
/*         not save the corresponding portion of the path.       */
/*                                                               */
/*****************************************************************/

int fnSplit(char *spec,             /* Original file spec        */
            char *drive,            /* Drive spec                */
            char *pname,            /* Path w/ drive spec        */
            char *path,             /* Path spec                 */
            char *fname,            /* File name + extension     */
            char *name,             /* File name                 */
            char *ext)              /* File extension            */
{
      int ret_code = 0;
      char *d = spec, *p, *e;

      unix2dos(spec);

      if (':' == spec[1])
      {
            if (drive)
                  strncpy(drive, spec, 2);
            drive[2] = NUL;
            d += 2;
            ret_code |= Drive_;
      }
      else
      {
            if (drive)
                  *drive = NUL;
      }

      if (NULL != (p = strrchr(d, '\\')))
```

```
101 |       {
102 |             char ch;
103 |
104 |             ch = *(++p);
105 |             *p = NUL;
106 |             if (path)
107 |                   strcpy(path, d);
108 |             if (pname)
109 |                   strcpy(pname, spec);
110 |             if (has_wild(d))
111 |                   ret_code |= Wildpath_;
112 |             *p = ch;
113 |             ret_code |= Directory_;
114 |       }
115 |       else
116 |       {
117 |             if (path)
118 |                   *path = NUL;
119 |             if (pname)
120 |             {
121 |                   if (drive)
122 |                         strcpy(pname, drive);
123 |                   else  *pname = NUL;
124 |             }
125 |             p = d;
126 |
127 |             if ('.' == *p)
128 |             {
129 |                   size_t dot_length;
130 |
131 |                   ret_code |= Directory_;
132 |                   for (dot_length = 0; '.' == p[dot_length]; ++dot_length)
133 |                         ;
134 |                   if (path)
135 |                   {
136 |                         strncat(path, p, dot_length);
137 |                         strcat(path, "\\");
138 |                   }
139 |                   if (pname)
140 |                   {
141 |                         strncat(pname, p, dot_length);
142 |                         strcat(pname, "\\");
143 |                   }
144 |                   if (fname)
145 |                         *fname = NUL;
146 |                   if (name)
147 |                         *name  = NUL;
148 |                   if (ext)
149 |                         *ext   = NUL;
150 |                   return ret_code;
151 |             }
152 |       }
153 |       if (fname)
154 |             strcpy (fname, p);
155 |       if (has_wild(p))
156 |             ret_code |= Wildname_;
157 |       if (*p)
158 |             ret_code |= Filename_;
159 |       if (NULL != (e = strrchr(p, '.')))
160 |       {
161 |             *e = NUL;
162 |             if (name)
163 |                   strcpy(name, p);
164 |             *e = '.';
165 |             if (ext)
166 |                   strcpy(ext, e);
167 |             ret_code |= Extension_;
168 |       }
169 |       else
170 |       {
171 |             if (name)
172 |                   strcpy(name,p);
173 |             if (ext)
174 |                   *ext = NUL;
175 |       }
176 |       return ret_code;
177 | }
178 |
179 | /*****************************************************************/
180 | /*                                                               */
181 | /*  fnMerge()                                                    */
182 | /*                                                               */
183 | /*  Creates file specification from component parts. Similar to  */
184 | /*  compiler-specific fnmerge() or _makepath().                  */
185 | /*                                                               */
186 | /*  Arguments 1 - Buffer to receive file specification           */
187 | /*            2 - drive specification                            */
188 | /*            3 - drive/path specification                       */
189 | /*            4 - path specification                             */
190 | /*            5 - name.ext specification                         */
191 | /*            6 - name specification                             */
192 | /*            7 - ext specification                              */
193 | /*                                                               */
194 | /*  Returns: Reassembled name                                    */
195 | /*                                                               */
196 | /*  Side Effects: None                                           */
197 | /*                                                               */
198 | /*****************************************************************/
199 |
200 | char *fnMerge(char *spec,                /* File spec buffer     */
```

```
201 |               char *drive,               /* Drive spec           */
202 |               char *pname,               /* Path w/ drive spec   */
203 |               char *path,                /* Path spec            */
204 |               char *fname,               /* File name + extension*/
205 |               char *name,                /* File name            */
206 |               char *ext)                 /* File extension       */
207 | {
208 |       *spec = NUL;
209 |
210 |       if (pname && *pname)
211 |             strcpy(spec, pname);
212 |       else
213 |       {
214 |             if (drive && *drive)
215 |                   strcpy(spec, drive);
216 |             if (path && *path)
217 |                   strcpy(spec, path);
218 |       }
219 |
220 |       unix2dos(spec);
221 |
222 |       if (*spec && '\\' != LAST_CHAR(spec) && (':' != LAST_CHAR(spec)))
223 |             strcat(spec, "\\");
224 |
225 |       if (fname && *fname)
226 |             strcat(spec, fname);
227 |       else
228 |       {
229 |             if (name && *name)
230 |                   strcat(spec, name);
231 |             else  return spec;
232 |             if (ext && *ext)
233 |             {
234 |                   if ('.' != *ext)
235 |                         strcat(spec, ".");
236 |                   strcat(spec, ext);
237 |             }
238 |       }
239 |       return strupr(spec);
240 | }
241 |
242 | #ifdef TEST
243 |
244 | #include <stdio.h>
245 |
246 | main(int argc, char *argv[])
247 | {
248 |       char pname[FILENAME_MAX], drive[3], path[FILENAME_MAX];
249 |       char fname[13], name[9], ext[4], fullname[FILENAME_MAX];
250 |
251 |       if (2 > argc)
252 |       {
253 |             puts("\aUsage: FNSTST [d:][path\\][file][.ext]");
254 |             abort();
255 |       }
256 |       printf("fnSplit(%s) returned \n", argv[1]);
257 |       printf("%X\n",
258 |             fnSplit(argv[1], drive, pname, path, fname, name, ext));
259 |       printf("drive: %s\n", drive);
260 |       printf("pname: %s\n", pname);
261 |       printf("path : %s\n", path);
262 |       printf("fname: %s\n", fname);
263 |       printf("name : %s\n", name);
264 |       printf("ext  : %s\n", ext);
265 |
266 |       printf("\nCalling fnMerge() to reassemble everything returned %s\n",
267 |              fnMerge(fullname, drive, pname, path, fname, name, ext));
268 | }
269 |
270 | #endif /* TEST */
```

```
TEXT STATISTICS

 10784 characters
   270 lines

LEXICAL STATISTICS

    92 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
    11 constants [character]
    15 constants [string]
    13 constants [numeric]

SYMBOL TABLE

    Boolean_T       36
    Directory_             113     131
    Drive_          92
    Extension_             167
    FILENAME_MAX           248+    249
    False_          40
    Filename_      158
    LAST_CHAR      222+
    NUL             90      97     105     118     123     145     147     149     161     174     208
    NULL            38+    100     159
    TEST           242
    True_           39
    Wildname_      156
    Wildpath_      111
    abort          254
    argc           246     251
    argv           246     256     258
    ch             102     104     112
    d       82      91     100     107     110     125
    dot_length             129     132+    136     141
    drive           74      88      89      90      96      97     121     122     201     214+    215
           248     258     259     267
    e       82     159     161     164     166
    ext             79     148     149     165     166     173     174     206     232+    234     236
           249     258     264     267
    fnMerge        200     267
    fnSplit         73     258
    fname           77     144     145     153     154     204     225+    226     249     258     262
           267
    fullname       249     267
    h       18      19     244
    has_wild        36     110     155
    main           246
    name            78     146     147     162     163     171     172     205     229+    230     249
           258     263     267
    p       82     100     104     105     112     125     127     132     136     141     154     155
           157     159     163     172
    path            76     106     107     117     118     134     136     137     203     216+    217
           248     258     261     267
    pname           36      38+     75     108     109     119     122     123     139     141     142
           202     210+    211     248     258     260     267
    printf         256     257     259     260     261     262     263     264     266
    puts           253
    ret_code        81      92     111     113     131     150     156     158     167     176
    size_t         129
    spec            73      82      84      86      89     109     200     208     211     215     217
           220     222+    223     226     230     231     235     236     239
    stdio          244
    stdlib          19
    strcat         137     142     223     226     230     235     236
    strchr          38+
    strcpy         107     109     122     154     163     166     172     211     215     217
    string          18
    strncat        136     141
    strncpy         89
    strrchr        100     159
    strupr         239
    unix2dos        84     220
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**          MFL.LIB  function source code
**  Copyright 1986, S.E. Margison
**
**  Modified 1989-93; Copyright Robert B.Stout dba MicroFirm as part of the
**  MicroFirm Function Library (MFL).
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  FUNCTION: fopenp, fopend, fopeng
**
**  Perform fopen() in path, stated environment, or both.
**  Current directory is always attempted first.
**  For fopeng() environment variable is searched before path
**  DRIVE and DIRECTORY strings may NOT be specified as part
**  of the filename.
*/

#include <stdio.h>
#include "snipfile.h"

FILE *fopenp(char *name, char *mode)
{
        char *newname;

        if (NULL != (newname = pexists(name)))
                return(fopen(newname, mode));
        else    return NULL;
}

FILE *fopend(char *name, char *mode, char *envar)
{
        char *newname;

        if (NULL != (newname = dexists(name, envar)))
                return(fopen(newname, mode));
        else    return NULL;
}

FILE *fopeng(char *name, char *mode, char *envar)
{
        char *newname;

        if (NULL != (newname = gexists(name, envar)))
                return(fopen(newname, mode));
        else    return NULL;
}
```

TEXT STATISTICS

```
  1594 characters
    51 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    FILE            26      35      44
    NULL            30      32      39      41      48      50
    dexists         39
    envar           35      39      44      48
    fopen           31      40      49
    fopend          35
    fopeng          44
    fopenp          26
    gexists         48
    h       23
    mode            26      31      35      40      44      49
    name            26      30      35      39      44      48
    newname         28      30      31      37      39      40      46      48      49
    pexists         30
    stdio           23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** fork.c (fork.h) - fork output to multiple file(s).
**
** By Dustin Puryear <dpuryear@delphi.com>
** Placed in the Public Domain.
*/

#include "fork.h"

int main(int argc, char *argv[])
{
      LL  files;     /* Linked-list of file pointers */
      int i;

      if ( argc < 2 )
      {
            help();
            exit(EXIT_FAILURE);
      }

      files.head = files.tail = NULL;

      /*
      ** First of all, find out if the user request help or if
      ** he disabled the console.
      */

      {
            int x;   /* Use as console toggle */

            for ( x = 0, i = 1; i < argc; i++ )
                  if ( argv[i][0] == '/' )
                        switch ( argv[i][1] )
                        {
                              case '?' :
                                    help();
                                    exit(EXIT_FAILURE);
                              case 'C' :
                              case 'c' :
                                    x = 1;
                                    break;
                        }

            /*
            ** x not toggled, therefore add stdout (console) to list.
            */

            if ( !x )
                  llopen(&files, stdout);
            else
            {
                  puts("Output to files only ...");
                  fflush(stdout);
            }

      }

      /*
      ** Go through command line again. Now look to open files. If
      ** you cannot open a file, ignore it!
      */

      {
            FILE  *fptr;              /* current file opened         */

            for ( i = 1; i < argc; i++ )
                  if ( argv[i][0] != '/' )
                  {
                        switch ( argv[i][0] )
                        {
                              case '*' :    /* Overwrite mode         */
                                    fptr = fopen(argv[i] + 1, "wb");
                                    break;
                              default  :    /* Append mode            */
                                    fptr = fopen(argv[i], "ab");
                                    break;
                        }
                        if ( fptr != NULL )
                              llopen(&files, fptr);
                  }

      }

      /*
      ** Get characters from stdin and place into a buffer. When buffer
      ** becomes full, write it out to all files in list.
      */

      {
            char  buffer[MAXBUFF]; /* buffer of chars      */
            int   i;               /* buffer index         */
            int   ch;              /* character from stdin */

            i = 0;
            while ( (ch = getchar()) != EOF )
            {
                  buffer[i++] = (char) ch;
                  if ( i == MAXBUFF )
```

```
101 |                   {
102 |                         output(&files, buffer, i - 1);
103 |                         i = 0;
104 |                   }
105 |             }
106 |
107 |             /*
108 |             ** If characters read, but have not been written out,
109 |             ** write them now.
110 |             */
111 |
112 |             if ( i != 0 && i < MAXBUFF )
113 |                   output(&files, buffer, i - 1);
114 |       }
115 |
116 |       /*
117 |       ** Close all files and remove list.
118 |       */
119 |
120 |       llfree(&files);
121 |
122 |       return (EXIT_SUCCESS);
123 | }
124 |
125 | /*
126 | ** llopen()
127 | **
128 | ** Place a new node at the end of the ll, containing the file pointer.
129 | */
130 |
131 | void llopen(LL *files, FILE *fptr)
132 | {
133 |       FILENODE *pnode;
134 |
135 |       pnode = (FILENODE *) malloc(sizeof(FILENODE));
136 |       if ( pnode == NULL )
137 |       {
138 |             perror("FORK ");
139 |             llfree(files);
140 |             exit(EXIT_FAILURE);
141 |       }
142 |       pnode->ptr = fptr;
143 |       pnode->next = NULL;
144 |
145 |       /*
146 |       ** If a ll does not exist, start a new one. Otherwise, continue
147 |       ** with the old.
148 |       */
149 |
150 |       if ( files->head == NULL )
151 |             files->head = files->tail = pnode;
152 |       else
153 |       {
154 |             files->tail->next = pnode;
155 |             files->tail = pnode;
156 |       }
157 | }
158 |
159 | /*
160 | ** llfree()
161 | **
162 | ** Free the memory consumed by the linked-list and closes any open files.
163 | */
164 |
165 | void llfree(LL *files)
166 | {
167 |       FILENODE *del;
168 |
169 |       while ( files->head != NULL )
170 |       {
171 |             fclose(files->head->ptr);
172 |             del = files->head;
173 |             files->head = files->head->next;
174 |             free(del);
175 |       }
176 | }
177 |
178 | /*
179 | ** output()
180 | **
181 | ** Output buffer to the file within the structure.
182 | */
183 |
184 | void output(LL *files, char *buffer, int size)
185 | {
186 |       int      i;
187 |       FILENODE *current;
188 |
189 |       /*
190 |       ** If console is still active, output to it first.
191 |       */
192 |
193 |       current = files->head;
194 |       if ( current->ptr == stdout )
195 |       {
196 |             for ( i = 0; i <= size; i++ )
197 |                   putchar(buffer[i]);
198 |             fflush(stdout);
199 |             current = current->next;
200 |       }
```

```
201 |
202 |       while ( current != NULL )
203 |       {
204 |             fwrite(buffer, sizeof(char) * size, 1, current->ptr);
205 |             current = current->next;
206 |       }
207 | }
208 |
209 | /*
210 | ** help()
211 | **
212 | ** Show a help screen to the user.
213 | */
214 |
215 | void help(void)
216 | {
217 |       putchar('\n');
218 |       printf("Fork - (pd) %s v%s Dustin Puryear\n", __DATE__, VER);
219 |       puts("This program is in Public Domain.");
220 |       putchar('\n');
221 |       puts("Fork piped output to the console and/or file(s).");
222 |       putchar('\n');
223 |       puts("[command |] fork [/?][/C] [filename.ext *filename.ext ...]");
224 |       putchar('\n');
225 |       puts("command       - Command that you wish to fork output.");
226 |       puts("/?            - Call this screen.");
227 |       puts("/C            - Do not output to console.");
228 |       puts("filename.ext  - Fork output to this file (append).");
229 |       puts("*filename.ext - Fork output to this file (overwrite).");
230 |       putchar('\n');
231 |       puts("NOTE: FORK is case insensitive.");
232 |       putchar('\n');
233 |
234 |       fflush(stdout);
235 | }
```

```
TEXT STATISTICS

  6021 characters
   235 lines

LEXICAL STATISTICS

    22 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
    12 constants [character]
    15 constants [string]
    17 constants [numeric]

SYMBOL TABLE

    EOF             97
    EXIT_FAILURE            20      39     140
    EXIT_SUCCESS           122
    FILE             66     131
    FILENODE        133     135+    167     187
    LL               14     131     165     184
    MAXBUFF          92     100     112
    NULL             23      80     136     143     150     169     202
    VER             218
    __DATE__        218
    argc             12      17      33      68
    argv             12      34      35      69      71      74      77
    buffer           92      99     102     113     184     197     204
    ch               94      97      99
    current         187     193     194     199+    202     204     205+
    del             167     172     174
    exit             20      39     140
    fclose          171
    fflush           55     198     234
    files            14      23+     51      81     102     113     120     131     139     150     151+
          154     155     165     169     171     172     173+    184     193
    fopen            74      77
    fptr             66      74      77      80      81     131     142
    free            174
    fwrite          204
    getchar          97
    head             23     150     151     169     171     172     173+    193
    help             19      38     215
    i        15      33+     34      35      68+     69      71      74      77      93      96      99
           100     102     103     112+    113     186     196+    197
    llfree          120     139     165
    llopen           51      81     131
    main             12
    malloc          135
    next            143     154     173     199     205
    output          102     113     184
    perror          138
    pnode           133     135     136     142     143     151     154     155
    printf          218
    ptr             142     171     194     204
    putchar         197     217     220     222     224     230     232
    puts             54     219     221     223     225     226     227     228     229     231
    size            184     196     204
    stdout           51      55     194     198     234
    tail             23     151     154     155
    x        31      33      42      50
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** fork.h -- get characters from a pipe and display to all files
**           needed (assumes stdout), Public Domain.
**
** PROGRAMMER : Dustin Puryear <dpuryear@delphi.com>
** COMPILER   : Mix PowerC v2.2.0a (MSDOS)
** HISTORY    : May 4, 1994  - Original
**              May 11, 1994 - Output in binary, buffer read of stdin.
**              June 8, 1994 - Cleaned up header, and removed some
**                             unneeded code.
*/

#include <stdio.h>    /* standard include file        */
#include <stdlib.h>   /* malloc(), free(), exit()     */
#include <string.h>   /* strcpy()                     */
#include <ctype.h>    /* tolower()                    */
#include <direct.h>   /* MAXPATH definition (MSDOS)   */

#ifndef __DATE__
#define __DATE__ "June 8 1994"    /* Date last updated             */
#endif
#ifndef MAXPATH
#define MAXPATH 80                /* Make sure there is a MAXPATH  */
#endif
#ifndef __STDC__
#ifndef EXIT_SUCCESS
#define EXIT_SUCCESS 0            /* Define exit values            */
#endif
#ifndef EXIT_FAILURE
#define EXIT_FAILURE 1
#endif
#endif
#define VER     "1.02"            /* Version number                */
#define MAXBUFF 4096              /* Maximum buffer size           */

typedef struct filenode
{
      FILE            *ptr;
      struct filenode *next;
} FILENODE;                 /* Node in ll containing a file pointer */
typedef struct
{
      FILENODE     *head;
      FILENODE     *tail;
} LL;                       /* ll of FILE pointers                */

void llopen(LL *files, FILE *fptr);
void llfree(LL *files);
void output(LL *files, char *buffer, int size);
void help(void);
```

```
TEXT STATISTICS

  1754 characters
    52 lines

LEXICAL STATISTICS

    14 comments [std-C]
     0 comments [C++]
    21 preprocessor instructions
     0 constants [character]
     2 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            31      32
    EXIT_SUCCESS            28      29
    FILE            40      49
    FILENODE        42      45      46
    LL              47      49      50      51
    MAXBUFF         36
    MAXPATH         24      25
    VER             35
    __DATE__        21      22
    __STDC__        27
    buffer          51
    ctype           18
    direct          19
    filenode        38      41
    files           49      50      51
    fptr            49
    h       15      16      17      18      19
    head            45
    help            52
    llfree          50
    llopen          49
    next            41
    output          51
    ptr             40
    size            51
    stdio           15
    stdlib          16
    string          17
    tail            46
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Writen by:  Wayne Halsdorf
**  Finished:   12-Jul-1994
**
**  This program is released to the PUBLIC DOMAIN.
**
**  A method for formatting floppy disks.
**
**  Testted with Microsoft C 7.0
*/

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#if defined(__TURBOC__)
 #include    <alloc.h>
#elif defined(__ZTC__)
 #if !defined(__SC__) || (__SC__ < 0x700)
  #error FORMAT.C >> _fcalloc() not supported bt ZTC or SC ver. 6.xx
 #endif
#else
 #include    <malloc.h>
#endif
#include    <dos.h>
#include    <direct.h>
#include    <time.h>
#include    <conio.h>
#include    <bios.h>
#include    <ctype.h>
#include    "format.h"


DPB        FAR        *dpb;
FPB        FAR        *fpb;
APB        FAR        *apb;
IOPB       FAR        *iopb;
BOOTSECTOR FAR        *BootSector;

FORMAT_TABLE    D360[] = {
      {"360K", 0, 1, 2, 40,  9, 7, 0xFD, 1, 3, 706},
      {"360K", 0, 1, 2, 40,  9, 7, 0xFD, 2, 2, 354},
      {"360K", 0, 1, 2, 40,  9, 5, 0xFD, 4, 1, 178}
};

FORMAT_TABLE    D1200[] = {
      {"360K", 0, 1, 2, 40,  9,  7, 0xFD, 1, 3,  706},
      {"360K", 0, 1, 2, 40,  9,  7, 0xFD, 2, 2,  354},
      {"360K", 0, 1, 2, 40,  9,  5, 0xFD, 4, 1,  178},
      {"1.2M", 1, 0, 2, 80, 15,  7, 0xF9, 1, 7, 2378},
      {"1.2M", 1, 0, 2, 80, 15,  7, 0xF9, 2, 4, 1192},
      {"1.2M", 1, 0, 2, 80, 15,  7, 0xF9, 4, 2,  597},
      {"1.2M", 1, 0, 2, 80, 15,  5, 0xF9, 8, 1,  299},
      {"1.2M", 1, 0, 2, 80, 15, 14, 0xF9, 1, 7, 2371},
      {"1.2M", 1, 0, 2, 80, 15, 15, 0xF9, 2, 4, 1188},
      {"1.2M", 1, 0, 2, 80, 15, 15, 0xF9, 4, 2,  595},
      {"1.2M", 1, 0, 2, 80, 15, 13, 0xF9, 8, 1,  298}
};

FORMAT_TABLE    D720[] = {
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 1, 5, 1422},
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 2, 3,  713},
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 4, 2,  357},
      { "720K", 2, 0, 2, 80,  9,  5, 0xF9, 8, 1,  179}
};

FORMAT_TABLE    D1440[] = {
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 1, 5, 1422},
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 2, 3,  713},
      { "720K", 2, 0, 2, 80,  9,  7, 0xF9, 4, 2,  357},
      { "720K", 2, 0, 2, 80,  9,  5, 0xF9, 8, 1,  179},
      {"1.44M", 7, 0, 2, 80, 18,  7, 0xF0, 1, 9, 2854},
      {"1.44M", 7, 0, 2, 80, 18,  7, 0xF0, 2, 5, 1431},
      {"1.44M", 7, 0, 2, 80, 18,  5, 0xF0, 4, 3,  717},
      {"1.44M", 7, 0, 2, 80, 18,  3, 0xF0, 8, 2,  359},
      {"1.44M", 7, 0, 2, 80, 18, 14, 0xF0, 1, 9, 2847},
      {"1.44M", 7, 0, 2, 80, 18, 13, 0xF0, 2, 5, 1428},
      {"1.44M", 7, 0, 2, 80, 18, 13, 0xF0, 4, 3,  715},
      {"1.44M", 7, 0, 2, 80, 18, 11, 0xF0, 8, 2,  358}
};

unsigned int    GetKey(void)
{
      while (!_bios_keybrd(_KEYBRD_READY))
            ;
      return(_bios_keybrd(_KEYBRD_READ) & 0xff);
}

void    GetLine(char FAR *in, unsigned int len)
{
      unsigned int    k, l;
      char FAR *p;

      p = in;
      l = 0;
      *p = '\0';
      do
      {
            k = GetKey();
```

```
101 |             if(k == 8 && l)
102 |             {
103 |                   l--;
104 |                   p--;
105 |                   *p = '\0';
106 |                   printf("\b \b");
107 |             }
108 |             else  if(__iscsym(k) && l < len)
109 |             {
110 |                   k = (unsigned int)toupper(k);
111 |                   *p = (char)k;
112 |                   l++;
113 |                   p++;
114 |                   *p = '\0';
115 |                   printf("%c", k);
116 |             }
117 |       } while (k != 0x0d);
118 | }
119 |
120 | void    InitVars(void)
121 | {
122 |       dpb = _fcalloc(1, sizeof(DPB));
123 |       fpb = _fcalloc(1, sizeof(FPB));
124 |       apb = _fcalloc(1, sizeof(APB));
125 |       BootSector = _fcalloc(1, sizeof(BOOTSECTOR));
126 | }
127 |
128 | void    FreeVars(void)
129 | {
130 |       if(apb)
131 |             _ffree(apb);
132 |       if(dpb)
133 |             _ffree(dpb);
134 |       if(fpb)
135 |             _ffree(fpb);
136 |       if(iopb->dta)
137 |             _ffree(iopb->dta);
138 |       if(iopb)
139 |             _ffree(iopb);
140 |       if(BootSector)
141 |             _ffree(BootSector);
142 | }
143 |
144 | void    Error_Message(char *message_)
145 | {
146 |       printf(message_);
147 |       FreeVars();
148 |       exit(1);
149 | }
150 |
151 | unsigned int    AvailableDrives(void)
152 | {
153 |       _outp(0x70, 16);
154 |       return(_inp(0x71));
155 | }
156 |
157 | void    SelectDrive(unsigned int *dr)
158 | {
159 |       unsigned int    d, a;
160 |
161 |       a = 0;
162 |       d = AvailableDrives();
163 |       if(d & 0x0f)            /* drive b */
164 |             a = 2;
165 |       if(d & 0xf0)            /* drive a */
166 |             a |= 1;
167 |
168 |       switch(a)
169 |       {
170 |       case 0:
171 |             Error_Message("No floppies found\n");
172 |             break;
173 |
174 |       case 1:
175 |             *dr = 1;
176 |             break;
177 |
178 |       case 2:
179 |             *dr = 2;
180 |             break;
181 |
182 |       case 3:
183 |             printf("Format which drive (A or B) : ");
184 |             do
185 |             {
186 |                   d = GetKey();
187 |                   d = (unsigned int)(toupper(d) - 'A');
188 |             } while (d > 1);
189 |             printf("%c\n", d + 'A');
190 |             *dr = d + 1;
191 |             break;
192 |       }
193 | }
194 |
195 | void    GetDrive(char *c_, unsigned int *dr)
196 | {
197 |       *dr = (unsigned int)(toupper(*c_) - 'A' + 1);
198 |       if(*dr < 1)
199 |             Error_Message("Attempted to format unknown device\n");
200 |
```

```
201 |       if(*dr > 2)
202 |             Error_Message("Attempted to format non-floppy device\n");
203 | }
204 |
205 | void    GetFloppyTable(unsigned int drive_number,
206 |                        unsigned int *s,
207 |                        FORMAT_TABLE **ft)
208 | {
209 |       unsigned int    d;
210 |
211 |       d = AvailableDrives();
212 |       if(drive_number == 1)
213 |             d = d >> 4;
214 |       else  d = d & 15;
215 |
216 |       switch(d)
217 |       {
218 |       case 1:
219 |             *ft = D360;
220 |             *s = sizeof(D360) / sizeof(FORMAT_TABLE);
221 |             break;
222 |
223 |       case 2:
224 |             *ft = D1200;
225 |             *s = sizeof(D1200) / sizeof(FORMAT_TABLE);
226 |             break;
227 |
228 |       case 3:
229 |             *ft = D720;
230 |             *s = sizeof(D720) / sizeof(FORMAT_TABLE);
231 |             break;
232 |
233 |       case 4:
234 |             *ft = D1440;
235 |             *s = sizeof(D1440) / sizeof(FORMAT_TABLE);
236 |             break;
237 |
238 |       default:
239 |             Error_Message("Don't know about formats for select drive\n");
240 |             break;
241 |       }
242 | }
243 |
244 | void    DriveParameters(unsigned int *format,
245 |                         FORMAT_TABLE *ftable,
246 |                         unsigned int s)
247 | {
248 |       unsigned int    x;
249 |
250 |       printf("Available formats for selected drive\n");
251 |       printf("    Format Directory Cluster  Clusters  Storage\n");
252 |       printf("     Type  Available  Size    Available Capacity\n");
253 |       for (x = 0; x < s; x++)
254 |       {
255 |             printf(" %c) %6s    %3u     %4u      %4u   %9lu\n", x + 'A',
256 |                    ftable[x].Formats_,
257 |                    ftable[x].Max_Entries * 16,
258 |                    ftable[x].Cluster_Size * 512,
259 |                    ftable[x].Available,
260 |                    512L * ftable[x].Available * ftable[x].Cluster_Size);
261 |       }
262 |       printf("Input which format: ");
263 |
264 |       do
265 |       {
266 |             x = GetKey();
267 |             x = (unsigned int)(toupper((int)x) - 'A');
268 |       } while (x >= s);
269 |
270 |       printf("%c\n", x + 'A');
271 |       *format = x;
272 | }
273 |
274 | unsigned int DOS_IOCTL(unsigned drive, unsigned func, void _far *data)
275 | {
276 |       union   REGS   ir, or;
277 |       struct  SREGS  sr;
278 |
279 |       ir.x.ax = GENERIC_IO;
280 |       ir.x.bx = drive;
281 |       ir.x.cx = func;
282 |       ir.x.dx = _FP_OFF(data);
283 |       sr.ds = _FP_SEG(data);
284 |       _int86x(0x21, &ir, &or, &sr);
285 |       return(or.x.cflag);
286 | }
287 |
288 | unsigned int Extended_Error_Code(void)
289 | {
290 |       union   _REGS   ir, or;
291 |
292 |       ir.h.ah = 0x59;
293 |       ir.x.bx = 0;
294 |       _int86(0x21, &ir, &or);
295 |       return(or.h.bh);
296 | }
297 |
298 | unsigned int SerialNumber(unsigned drive, unsigned func, void _far *data)
299 | {
300 |       union   _REGS   ir, or;
```

```
301 |       struct  _SREGS  sr;
302 |
303 |       ir.x.ax = SERIAL_NUMBER + func;
304 |       ir.x.bx = drive;
305 |       ir.x.dx = _FP_OFF(data);
306 |       sr.ds = _FP_SEG(data);
307 |       _int86x(0x21, &ir, &or, &sr);
308 |       if(or.x.cflag)
309 |             return(or.x.ax);
310 |       else  return(0);
311 | }
312 |
313 | unsigned long GenSerialNumber(void)
314 | {
315 |       union   _REGS   ir, or;
316 |       union {
317 |             unsigned long   d;
318 |             unsigned int    i[2];
319 |       } g;
320 |
321 |       ir.x.ax = GetDate;
322 |       _int86(0x21, &ir, &or);
323 |       g.i[0] = or.x.cx;
324 |       g.i[1] = or.x.dx;
325 |       ir.x.ax = GetTime;
326 |       _int86(0x21, &ir, &or);
327 |       g.i[0] += or.x.cx;
328 |       g.i[1] += or.x.dx;
329 |       return(g.d);
330 | }
331 |
332 | unsigned int    FlopyStatus(unsigned int drive)
333 | {
334 |       union   _REGS   ir, or;
335 |
336 |       ir.h.ah = 1;
337 |       ir.x.dx = drive - 1;
338 |       _int86(0x13, &ir, &or);
339 |       return(or.h.al);
340 | }
341 |
342 | void    ResetDisk(unsigned int drive)
343 | {
344 |       union   _REGS   ir, or;
345 |
346 |       ir.h.ah = 0;
347 |       ir.x.dx = drive - 1;
348 |       _int86(0x13, &ir, &or);
349 | }
350 |
351 | /*
352 | ** gain acces to drive (required if disk in drive is unformated)
353 | */
354 |
355 | void    SetAccess(unsigned int drivenumber, APB FAR *apb)
356 | {
357 |       apb->Function = 0;
358 |       DOS_IOCTL(drivenumber, GENERIC_GETACC, apb);
359 |       if(apb->Flag == 0)
360 |       {
361 |             apb->Flag = 1;
362 |             DOS_IOCTL(drivenumber, GENERIC_SETACC, apb);
363 |       }
364 | }
365 |
366 | void    InitParameters(unsigned int drive_number,
367 |                        unsigned int format,
368 |                        FORMAT_TABLE *ftable)
369 | {
370 |       unsigned int    t;
371 |
372 |       dpb->Function = 0;
373 |       dpb->Device_Type = ftable[format].Device_Type;
374 |       dpb->Tracks = ftable[format].Tracks;
375 |       dpb->Media_Type = ftable[format].Media_Type;
376 |       dpb->bpb.Bytes_Sector = 0x0200;
377 |       dpb->bpb.Cluster_Size = ftable[format].Cluster_Size;
378 |       dpb->bpb.Reserved_Sectors = 1;
379 |       dpb->bpb.Number_FATS = 2;
380 |       dpb->bpb.Max_Root_Entries = ftable[format].Max_Entries * 16;
381 |       dpb->bpb.Number_Sectors   = ftable[format].Tracks *
382 |                                   ftable[format].Heads *
383 |                                   ftable[format].Sectors_Per_Tracks;
384 |       dpb->bpb.Media_Descriptor = ftable[format].Media_Descriptor;
385 |       dpb->bpb.Sectors_FAT = ftable[format].FatSize;
386 |       dpb->bpb.Sectors_Track = ftable[format].Sectors_Per_Tracks;
387 |       dpb->bpb.Number_Heads = ftable[format].Heads;
388 |       dpb->bpb.Hidden_Sectors = 0;
389 |       dpb->bpb.Large_Number_Sectors = 0;
390 |       dpb->track.Number_Sectors = ftable[format].Sectors_Per_Tracks;
391 |       for (t = 0; t < dpb->track.Number_Sectors; t++)
392 |       {
393 |             dpb->track.track_layout[t].Sector_Number = t + 1;
394 |             dpb->track.track_layout[t].Sector_Size = dpb->bpb.Bytes_Sector;
395 |       }
396 |
397 |       iopb = _fcalloc(1, sizeof(IOPB));
398 |       iopb->dta = _fcalloc(dpb->bpb.Sectors_Track * dpb->bpb.Bytes_Sector,
399 |                            sizeof(char));
400 |       DOS_IOCTL(drive_number, GENERIC_SETDEV, dpb);
```

```
401 |       dpb->Function = 5;
402 | }
403 |
404 | #ifdef  _MSC_VER
405 | #pragma warning(disable:4001)
406 | #endif
407 |
408 | /*
409 | ** format each track, both sides
410 | */
411 |
412 | void    FormatDisk(unsigned int drive_number,
413 |                    unsigned int ftype,
414 |                    FORMAT_TABLE *ftable)
415 | {
416 |       unsigned int    retries, er;
417 |
418 |       printf("Insert disk in drive %c: and press enter ",
419 |              drive_number + 'A' - 1);
420 |
421 |       while (GetKey() != 0x0d)
422 |             ;
423 |
424 |       printf("\nFormatting %s Data Disk, Please Wait.\n",
425 |              ftable[ftype].Formats_);
426 |       fpb->Function = 0;      /* format/Verify track */
427 |       for (fpb->Track = 0; fpb->Track < dpb->Tracks; fpb->Track += 1)
428 |       {
429 |             for (fpb->Head = 0;
430 |                  fpb->Head < ftable[ftype].Heads;
431 |                  fpb->Head += 1)
432 |             {
433 |                   SetAccess(drive_number, apb);
434 |                   DOS_IOCTL(drive_number, GENERIC_SETDEV, dpb);
435 |                   printf("Formating Track %2u Side %u\r",
436 |                          fpb->Track, fpb->Head);
437 |                   retries = 0;
438 |                   while (retries++ < 4 && DOS_IOCTL(drive_number,
439 |                         GENERIC_FORMAT, fpb))
440 |                   {
441 |                         er = FlopyStatus(drive_number);
442 |                         if((er & 0x80) == 0)
443 |                         {
444 |                               er = Extended_Error_Code();
445 |                               if(er == 0x0d)
446 |                                     Error_Message("Invlaid Media\n");
447 |
448 |                               if(er == 0x0b)
449 |                               {
450 |                                     printf("Disk write proteced.\n"
451 |                                            "Press enter after correcting "
452 |                                            "or escape to abort\n");
453 |                                     do
454 |                                     {
455 |                                           er = GetKey();
456 |                                           if(er == 0x1b)
457 |                                                 Error_Message("Format aborted"
458 |                                                       ". Disk now invalid.\n");
459 |                                     } while (er != 0x0d);
460 |                               }
461 |                         }
462 |                         ResetDisk(drive_number);
463 |                         SetAccess(drive_number, apb);
464 |                         DOS_IOCTL(drive_number, GENERIC_SETDEV, dpb);
465 |                   }
466 |             }
467 |       }
468 | }
469 |
470 | void    InitializeDisk(unsigned int drive_number)
471 | {
472 |       struct tm *today;
473 |       VPB     FAR   *vpb;
474 |       time_t   timer;
475 |       unsigned int    er;
476 |       char    *er_;
477 |       static char FAR iname[NLEN + 1];
478 |       static unsigned char FAR boot_start[BJLEN] = {0xeb,0x3C,0x90};
479 |       static unsigned char FAR os_[OLEN] = {"PDDF 1.0"};
480 |       static unsigned char FAR fat_start[3] = {0xfd,0xff,0xff};
481 |       static unsigned char FAR name[NLEN] = {"NO NAME    "};
482 |       static unsigned char FAR fatname[FLEN] = {"FAT12   "};
483 |       static unsigned char FAR boot_code[BLEN];
484 |       static unsigned char boot_code_default[] = {
485 |             0xfa,                     /* 7c3e cli                     */
486 |             0xbc, 0x00, 0x7c,         /* 7c3f mov sp, 7c00h           */
487 |             0xfb,                     /* 7c42 sti                     */
488 |             0xb2, 0x00,               /* 7c43 mov dl,0                */
489 |             0x33, 0xc0,               /* 7c45 xor ax,ax               */
490 |             0xcd, 0x13,               /* 7c47 int 13h                 */
491 |             0x0e,                     /* 7c49 push cs                 */
492 |             0x1f,                     /* 7c4a pop  ds                 */
493 |             0xfc,                     /* 7c4b cld                     */
494 |             0xbe, 0x63, 0x7c,         /* 7c4c mov si, OFFSET message  */
495 |             0xac,                     /* 7c4f lodsb                   */
496 |             0x0a, 0xc0,               /* 7c50 or  al,al               */
497 |             0x74, 0x09,               /* 7c52 je  7c5c                */
498 |             0xb4, 0x0e,               /* 7c54 mov ah,0eh              */
499 |             0xbb, 0x07, 0x00,         /* 7c56 mov bx,7                */
500 |             0xcd, 0x10,               /* 7c59 int 10h                 */
```

```
501 |             0xeb, 0xf2,             /* 7c5b jmp SHORT 7c50          */
502 |             0x33, 0xc0,             /* 7c5d xor ax,ax               */
503 |             0xcd, 0x16,             /* 7c5f int 16h                 */
504 |             0xcd, 0x19              /* 7c61 int 19h                 */
505 |                                     /* 7c63 db  'message'           */
506 |     };
507 |     static char boot_text_default[] = {
508 |             "\r\nNon-System disk for DATA USE ONLY!"
509 |             "\r\nProduced by a Public Domain Disk Formater 1.0 ."
510 |             "\r\nSource code freely available."
511 |             "\r\nReplace or remove disk then press any key when ready\r\n"
512 |       };
513 |
514 |       printf("\nInitializing Disk\n");
515 |       _fmemcpy(&boot_code[0], boot_code_default, sizeof(boot_code_default));
516 |       _fmemcpy(&boot_code[sizeof(boot_code_default)], boot_text_default,
517 |                 sizeof(boot_text_default));
518 |
519 |       /*
520 |       ** test to see if BOOT FATS and DIRECTORY records can be placed
521 |       */
522 |
523 |       iopb->Function = 0;
524 |       iopb->Side = 0;
525 |       iopb->Track = 0;
526 |       iopb->FirstSector = 0;
527 |       iopb->NumberSectors = dpb->bpb.Sectors_Track;
528 |       er = DOS_IOCTL(drive_number, GENERIC_READ, iopb);
529 |       _fmemset(iopb->dta, 0, dpb->bpb.Sectors_Track * dpb->bpb.Bytes_Sector);
530 |       er = DOS_IOCTL(drive_number, GENERIC_WRITE, iopb);
531 |       er = DOS_IOCTL(drive_number, GENERIC_VERIFY, iopb);
532 |       if(er)
533 |             Error_Message("Track 0 side 0 defective\n");
534 |
535 |       iopb->Function = 0;
536 |       iopb->Side = 1;
537 |       iopb->Track = 0;
538 |       iopb->FirstSector = 0;
539 |       iopb->NumberSectors = dpb->bpb.Sectors_Track;
540 |       er = DOS_IOCTL(drive_number, GENERIC_WRITE, iopb);
541 |       er = DOS_IOCTL(drive_number, GENERIC_VERIFY, iopb);
542 |       if(er)
543 |             Error_Message("Track 0 side 1 defective\n");
544 |
545 |       /*
546 |       ** place BOOT Record
547 |       */
548 |
549 |       iopb->Function = 0;
550 |       iopb->Side = 0;
551 |       iopb->Track = 0;
552 |       iopb->FirstSector = 0;
553 |       iopb->NumberSectors = 1;
554 |       _fmemcpy(BootSector->boot_start_, boot_start, BJLEN);
555 |       _fmemcpy(BootSector->os_name_, os_, OLEN);
556 |       _fmemcpy(&(BootSector->bpb), &(dpb->bpb), sizeof(BPB) + 1);
557 |       BootSector->bpb.Reserved[2] = 0x29;       /* needed for serial number */
558 |       _fmemcpy(BootSector->volname_, name, NLEN);
559 |       _fmemcpy(BootSector->fat_, fatname, FLEN);
560 |       _fmemcpy(BootSector->boot_code_, boot_code, BLEN);
561 |       _fmemcpy(iopb->dta, BootSector, sizeof(BOOTSECTOR));
562 |       er = DOS_IOCTL(drive_number, GENERIC_WRITE, iopb);
563 |       er = DOS_IOCTL(drive_number, GENERIC_VERIFY, iopb);
564 |       if(er)
565 |             Error_Message("Bad Boot Sector\n");
566 |
567 |       /*
568 |       ** place first FAT
569 |       */
570 |
571 |       _fmemset(iopb->dta, 0, dpb->bpb.Sectors_FAT * dpb->bpb.Bytes_Sector);
572 |       _fmemcpy(iopb->dta, fat_start, sizeof(fat_start));
573 |       iopb->Function = 0;
574 |       iopb->Side = 0;
575 |       iopb->Track = 0;
576 |       iopb->FirstSector = 1;
577 |       iopb->NumberSectors = dpb->bpb.Sectors_FAT;
578 |       er = DOS_IOCTL(drive_number, GENERIC_WRITE, iopb);
579 |       er = DOS_IOCTL(drive_number, GENERIC_VERIFY, iopb);
580 |       if(er)
581 |             Error_Message("Bad File Allocation Table\n");
582 |
583 |       /*
584 |       ** place second FAT
585 |       */
586 |
587 |       iopb->Function = 0;
588 |       iopb->Side = 0;
589 |       iopb->Track = 0;
590 |       iopb->FirstSector = 1 + dpb->bpb.Sectors_FAT;
591 |       iopb->NumberSectors = dpb->bpb.Sectors_FAT;
592 |       er = DOS_IOCTL(drive_number, GENERIC_WRITE, iopb);
593 |       er = DOS_IOCTL(drive_number, GENERIC_VERIFY, iopb);
594 |       if(er)
595 |             Error_Message("Bad File Allocation Table\n");
596 |
597 |       /*
598 |       ** place extend information on BOOT record
599 |       */
600 |
```

```
601 |       vpb = _fcalloc(1, sizeof(VPB));
602 |       SerialNumber(drive_number, SERIAL_READ, vpb);
603 |       today = localtime( &timer );
604 |       vpb->SerialNumber = GenSerialNumber();
605 |       printf("Internal Volume Name: ");
606 |       GetLine(iname, NLEN);
607 |       if(_fstrlen(iname))
608 |       {
609 |             _fmemset(vpb->VolumeLabel, ' ', NLEN);
610 |             _fmemcpy(vpb->VolumeLabel, iname, _fstrlen(iname));
611 |       }
612 |       er = SerialNumber(drive_number, SERIAL_WRITE, vpb);
613 |       _ffree(vpb);
614 |       if(er)
615 |       {
616 |             er = Extended_Error_Code();
617 |             switch(er)
618 |             {
619 |             case 0x15:
620 |                   er_ = "Drive not Ready. Can't update Boot.\n";
621 |                   break;
622 |
623 |             case 0x1F:
624 |                   er_ = "General Failure. Can't update Boot\n";
625 |                   break;
626 |
627 |             default:
628 |                   er_ = "Can't update Boot\n";
629 |                   break;
630 |             }
631 |             Error_Message(er_);
632 |       }
633 | }
634 |
635 | void    CheckDrive(unsigned int drive_number)
636 | {
637 |       union   _REGS   ir, or;
638 |       struct  _SREGS  sr;
639 |
640 |       ir.x.ax = CHECK_REMOVABLE;
641 |       ir.x.bx = drive_number;
642 |       _int86x(0x21, &ir, &or, &sr);
643 |       if(or.x.cflag)
644 |             Error_Message("Can't determine if media is removable.\n");
645 |
646 |       if(or.x.ax)
647 |             Error_Message("Selected media not removable, "
648 |                         "my not be a floppy.\n");
649 |
650 |       ir.x.ax = CHECK_REMOTE;
651 |       _int86x(0x21, &ir, &or, &sr);
652 |       if(or.x.cflag)
653 |             Error_Message("Can't determine if media is remote or shared.\n");
654 |
655 |       if((or.x.dx & 0x8000) != 0)
656 |             Error_Message("Can't format SUBSTITUTE drive.\n");
657 |
658 |       if((or.x.dx & 0x1000) != 0)
659 |             Error_Message("Can't format REMOTE drive.\n");
660 | }
661 |
662 | /*
663 | ** get real dos version
664 | */
665 |
666 | void    CheckDosVersion(void)
667 | {
668 |       if(_osmajor < 4)
669 |             Error_Message("Must be DOS 4.0 or higher.\n");
670 | }
671 |
672 | int     main(int argc, char *argv[])
673 | {
674 |       unsigned int    format, drive_number, size;
675 |       FORMAT_TABLE    *ftable;
676 |
677 |       CheckDosVersion();
678 |       InitVars();
679 |
680 |       if(argc == 1)
681 |             SelectDrive(&drive_number);
682 |       else  GetDrive(argv[1], &drive_number);
683 |
684 |       CheckDrive(drive_number);
685 |       GetFloppyTable(drive_number, &size, &ftable);
686 |       DriveParameters(&format, ftable, size);
687 |       InitParameters(drive_number, format, ftable);
688 |       FormatDisk(drive_number, format, ftable);
689 |       InitializeDisk(drive_number);
690 |       FreeVars();
691 |       return(0);
692 | }
```

```
TEXT STATISTICS

 21330 characters
   692 lines

LEXICAL STATISTICS

    35 comments [std-C]
     0 comments [C++]
    22 preprocessor instructions
    11 constants [character]
    78 constants [string]
   475 constants [numeric]

SYMBOL TABLE

    APB             37     124     355
    Available      259     260
    AvailableDrives         151     162     211
    BJLEN          478     554
    BLEN           483     560
    BOOTSECTOR              39     125     561
    BPB            556
    BootSector              39     125     140     141     554     555     556     557     558     559
           560     561
    Bytes_Sector            376     394     398     529     571
    C       21
    CHECK_REMOTE            650
    CHECK_REMOVABLE         640
    CheckDosVersion         666     677
    CheckDrive              635     684
    Cluster_Size            258     260     377+
    D1200           47      224     225
    D1440           68      234     235
    D360            41      219     220
    D720            61      229     230
    DOS_IOCTL      274     358     362     400     434     438     464     528     530     531     540
           541     562     563     578     579     592     593
    DPB             35      122
    Device_Type             373+
    DriveParameters         244     686
    Error_Message           144     171     199     202     239     446     457     533     543     565
           581     595     631     644     647     653     656     659     669
    Extended_Error_Code             288     444     616
    FAR             35      36      37      38      39      90      93     355     473     477     478
           479     480     481     482     483
    FLEN           482     559
    FORMAT          21
    FORMAT_TABLE            41      47      61      68     207     220     225     230     235     245
           368     414     675
    FPB             36      123
    FatSize        385
    FirstSector             526     538     552     576     590
    Flag           359     361
    FlopyStatus             332     441
    FormatDisk              412     688
    Formats_       256     425
    FreeVars       128     147     690
    Function       357     372     401     426     523     535     549     573     587
    GENERIC_FORMAT          439
    GENERIC_GETACC          358
    GENERIC_IO              279
    GENERIC_READ            528
    GENERIC_SETACC          362
    GENERIC_SETDEV          400     434     464
    GENERIC_VERIFY          531     541     563     579     593
    GENERIC_WRITE           530     540     562     578     592
    GenSerialNumber         313     604
    GetDate        321
    GetDrive       195     682
    GetFloppyTable          205     685
    GetKey          83     100     186     266     421     455
    GetLine         90     606
    GetTime        325
    Head           429     430     431     436
    Heads          382     387     430
    Hidden_Sectors          388
    IOPB            38     397
    InitParameters          366     687
    InitVars       120     678
    InitializeDisk          470     689
    Large_Number_Sectors            389
    Max_Entries             257     380
    Max_Root_Entries        380
    Media_Descriptor        384+
    Media_Type              375+
    NLEN           477     481     558     606     609
    NumberSectors           527     539     553     577     591
    Number_FATS             379
    Number_Heads            387
    Number_Sectors          381     390     391
    OLEN           479     555
    REGS           276
    Reserved       557
    Reserved_Sectors        378
    ResetDisk      342     462
    SC              21
    SERIAL_NUMBER           303
    SERIAL_READ             602
    SERIAL_WRITE            612
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Data structures and discriptions for IOCTL comes from the following
** MS-DOS Functions ISBN 1-55615-128-4
** Ralf Brown's Interupt List
*/

#ifndef FORMAT__H
#define FORMAT__H

#include "extkword.h"          /* For FAR  */

#ifdef  _MSC_VER
 #pragma warning(disable:4103)
#endif

#define MAX_SECTORS 18

#pragma pack(1)

#define IOCTL               0x4400
#define IOCTL_REMOVABLE     0x08
#define IOCTL_REMOTE        0x09
#define IOCTL_BLOCK         0x0D
#define SetDeviceParameters 0x40
#define WriteTrack          0x41
#define FormatVerifyTrack   0x42
#define SetAccessFlag       0x47        /* dos 4.0+ */
#define GetDeviceParameters 0x60
#define ReadTrack           0x61
#define VerifyTrack         0x62
#define GetAccessFlag       0x67        /* dos 4.0+ */
#define DiskDrive           0x0800

#define CHECK_REMOVABLE (IOCTL + IOCTL_REMOVABLE)
#define CHECK_REMOTE    (IOCTL + IOCTL_REMOTE)
#define GENERIC_IO      (IOCTL + IOCTL_BLOCK)
#define GENERIC_GETDEV  (DiskDrive + GetDeviceParameters)
#define GENERIC_SETDEV  (DiskDrive + SetDeviceParameters)
#define GENERIC_READ    (DiskDrive + ReadTrack)
#define GENERIC_WRITE   (DiskDrive + WriteTrack)
#define GENERIC_VERIFY  (DiskDrive + VerifyTrack)
#define GENERIC_FORMAT  (DiskDrive + FormatVerifyTrack)
#define GENERIC_GETACC  (DiskDrive + GetAccessFlag)
#define GENERIC_SETACC  (DiskDrive + SetAccessFlag)

#define SERIAL_NUMBER       0x6900
#define GetDate             0x2A00
#define GetTime             0x2C00
#define SERIAL_READ             0
#define SERIAL_WRITE            1

typedef struct
    {
    char            Formats_[6];
    unsigned char   Device_Type;
    unsigned char   Media_Type;
    unsigned int    Heads;
    unsigned int    Tracks;
    unsigned int    Sectors_Per_Tracks;
    unsigned int    Max_Entries;        /* in sectors */
    unsigned char   Media_Descriptor;
    unsigned char   Cluster_Size;
    unsigned int    FatSize;
    unsigned int    Available;
    } FORMAT_TABLE;

/*
**  Bios Parameter Block:
**      Function 44
**      Sub-function 0D
**      minor code 40 and 60
*/

typedef struct
    {
    unsigned int    Bytes_Sector;
    unsigned char   Cluster_Size;
    unsigned int    Reserved_Sectors;
    unsigned char   Number_FATS;
    unsigned int    Max_Root_Entries;
    unsigned int    Number_Sectors;
    unsigned char   Media_Descriptor;
    unsigned int    Sectors_FAT;
    unsigned int    Sectors_Track;
    unsigned int    Number_Heads;
    unsigned long   Hidden_Sectors;
    unsigned long   Large_Number_Sectors;   /* if Number_Sectors == 0 */
    unsigned char   Reserved[6];
    } BPB;

/*
**  Device Parameter Block:
**      Function 44
**      Sub-function 0D
**      minor codes 40 and 60
*/

typedef struct
```

```
    {
    unsigned char   Function;
    unsigned char   Device_Type;
    unsigned int    Device_Attribute;
    unsigned int    Tracks;
    unsigned char   Media_Type;
    BPB             bpb;
    struct TL
        {
        unsigned int    Number_Sectors;
        struct SECID
            {
            unsigned int    Sector_Number;
            unsigned int    Sector_Size;
            } track_layout[MAX_SECTORS];
        } track;
    } DPB;

/*
**  Format Parameter Block:
**      Function 44
**      Sub-function 0D,
**      minor code 42 and 62
*/

typedef struct
    {
    unsigned char   Function;
    unsigned int    Head;
    unsigned int    Track;
    } FPB;

/*
**  Volume Serial Parameter Block:
**      Function 44
**      Sub-function 0D,
**      minor code 46 and 66
*/

typedef struct
    {
    unsigned int    Function;           /* should be 0              */
    unsigned long   SerialNumber;       /* binary                   */
    unsigned char   VolumeLabel[11];    /* or "NO NAME    "         */
    unsigned char   FileSystemType[8];  /* "FAT12   " or "FAT16   " */
    } VPB;

/*
**  Access Parameter Block:
**      Function 44
**      Sub-function 0D
**      minor code 47 and 67
*/

typedef struct
    {
    unsigned char   Function;
    unsigned char   Flag;
    } APB;

/*
**  IO Parameter Block:
**      Function 44
**      Sub-function 0D
**      minor code 41, 61 and 62
*/

typedef struct
    {
    unsigned char       Function;
    unsigned int        Side;
    unsigned int        Track;
    unsigned int        FirstSector;
    unsigned int        NumberSectors;
    unsigned char FAR  *dta;
    } IOPB;


/* boot sector description */

#define BJLEN   3
#define OLEN    8
#define BREV    1
#define NLEN    11
#define FLEN    8
#define BLEN    512 - (FLEN + NLEN + BREV + OLEN + BJLEN + sizeof(BPB))

typedef struct
    {
    unsigned char   boot_start_[BJLEN];
    unsigned char   os_name_[OLEN];
    BPB             bpb;
    unsigned char   rev_[BREV];
    unsigned char   volname_[NLEN];
    unsigned char   fat_[FLEN];
    unsigned char   boot_code_[BLEN];
    } BOOTSECTOR;

#endif /* FORMAT__H */
```

```
TEXT STATISTICS

  5110 characters
   199 lines

LEXICAL STATISTICS

    19 comments [std-C]
     0 comments [C++]
    44 preprocessor instructions
     0 constants [character]
     1 constants [string]
    31 constants [numeric]

SYMBOL TABLE

    APB             159
    Available        66
    BJLEN           181     186     190
    BLEN            186     196
    BOOTSECTOR              197
    BPB              91     107     186     192
    BREV            183     186     193
    Bytes_Sector             78
    CHECK_REMOTE             37
    CHECK_REMOVABLE          36
    Cluster_Size             64      79
    DPB             117
    Device_Attribute        104
    Device_Type              57     103
    DiskDrive        34      39      40      41      42      43      44      45      46
    FAR             175
    FLEN            185     186     195
    FORMAT_TABLE             67
    FORMAT__H         9      10
    FPB             131
    FatSize          65
    FileSystemType          145
    FirstSector             173
    Flag            158
    FormatVerifyTrack        28      44
    Formats_         56
    Function        102     128     142     157     170
    GENERIC_FORMAT           44
    GENERIC_GETACC           45
    GENERIC_GETDEV           39
    GENERIC_IO               38
    GENERIC_READ             41
    GENERIC_SETACC           46
    GENERIC_SETDEV           40
    GENERIC_VERIFY           43
    GENERIC_WRITE            42
    GetAccessFlag            33      45
    GetDate          49
    GetDeviceParameters              30      39
    GetTime          50
    Head            129
    Heads            59
    Hidden_Sectors           88
    IOCTL            22      36      37      38
    IOCTL_BLOCK              25      38
    IOCTL_REMOTE             24      37
    IOCTL_REMOVABLE          23      36
    IOPB            176
    Large_Number_Sectors             89
    MAX_SECTORS              18     115
    Max_Entries              62
    Max_Root_Entries         82
    Media_Descriptor         63      84
    Media_Type               58     106
    NLEN            184     186     194
    NumberSectors           174
    Number_FATS              81
    Number_Heads             87
    Number_Sectors           83     110
    OLEN            182     186     191
    ReadTrack        31      41
    Reserved         90
    Reserved_Sectors         80
    SECID           111
    SERIAL_NUMBER            48
    SERIAL_READ              51
    SERIAL_WRITE             52
    Sector_Number           113
    Sector_Size             114
    Sectors_FAT              85
    Sectors_Per_Tracks               61
    Sectors_Track            86
    SerialNumber            143
    SetAccessFlag            29      46
    SetDeviceParameters              26      40
    Side            171
    TL              108
    Track           130     172
    Tracks           60     105
    VPB             146
    VerifyTrack              32      43
    VolumeLabel             144
    WriteTrack               27      42
    _MSC_VER         14
```

```
boot_code_              196
boot_start_             190
bpb             107     192
disable          15
dta             175
fat_            195
os_name_        191
pack             20
rev_            193
track           116
track_layout            115
volname_        194
warning          15
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FPSWITCH.C - Demonstrates using function pointers in lieu of switches
*/

#include <stdlib.h>     /* for NULL */

/*  Declare your functions here     */

char *cpfunc1(int);
char *cpfunc2(int);
char *cpfunc3(int);

void vfunc1(void);
void vfunc2(void);
void vfunc3(void);
void vfunc4(void);

/*
**  Old ways using switch statements
*/

char *oldcpswitch(int select, int arg)
{
      switch (select)
      {
      case 1:
            return cpfunc1(arg);

      case 2:
            return cpfunc2(arg);

      case 3:
            return cpfunc3(arg);

      default:
            return NULL;
      }
}

void oldvswitch(int select)
{
      switch (select)
      {
      case 1:
            vfunc1();
            break;

      case 2:
            vfunc2();
            break;

      case 3:
            vfunc3();
            break;

      case 4:
            vfunc4();
            break;
      }
}

/*
**  Using function pointers
*/

char *newcpswitch(int select, int arg)
{
      char *(*cpfunc[3])(int) = { cpfunc1, cpfunc2, cpfunc3 };

      if (select < 1 || select > 3)
            return NULL;
      return (*cpfunc[select-1])(arg);
}

void newvswitch(int select)
{
      void (*vfunc[4])(void) = { vfunc1, vfunc2, vfunc3, vfunc4 };

      if (select > 0 && select < 5)
            (*vfunc[select-1])();
}
```

TEXT STATISTICS

```
  1483 characters
    83 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     0 constants [string]
    15 constants [numeric]
```

SYMBOL TABLE

```
    NULL            38      73
    arg             24      29      32      35      68      74
    cpfunc          70      74
    cpfunc1         11      29      70
    cpfunc2         12      32      70
    cpfunc3         13      35      70
    h        7
    newcpswitch             68
    newvswitch              77
    oldcpswitch             24
    oldvswitch              42
    select          24      26      42      44      68      72+     74      77      81+     82
    stdlib           7
    vfunc           79      82
    vfunc1          15      47      79
    vfunc2          16      51      79
    vfunc3          17      55      79
    vfunc4          18      59      79
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  FPTR_ADD.C
  5 | **
  6 | **  Add any add any value to a far pointer and returns the result as a
  7 | **  normalized far pointer.
  8 | **
  9 | **  Public Domain by Soleil Lapierre
 10 | */
 11 |
 12 | #include "snpdosys.h"
 13 | #include "mk_fp.h"
 14 |
 15 |
 16 | void FAR *addptr (char FAR *p, unsigned long num)
 17 | {
 18 |       unsigned seg,off;
 19 |
 20 |       seg = FP_SEG(p); off = FP_OFF(p);
 21 |       seg += off>>4;   off &= 0x000F;
 22 |
 23 |       off += (unsigned)(num&0x0000000fL);
 24 |
 25 |       seg += off>>4;   off &= 0x000F;
 26 |       seg += (unsigned)num>>4;
 27 |
 28 |       return(MK_FP(seg,off));
 29 | }
 30 |
 31 | /*
 32 | **  Normalize a far pointer
 33 | */
 34 |
 35 | void FAR *farnormal(void FAR *ptr)
 36 | {
 37 |       unsigned long base, para;
 38 |
 39 |       base = ((unsigned long)(ptr) & 0xffff000fL);
 40 |       para = ((unsigned long)(ptr) & 0x0000fff0L);
 41 |       para <<= 12;
 42 |       return (void FAR *)(base + para);
 43 | }
```

TEXT STATISTICS

```
   863 characters
    43 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     2 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    FAR             16+     35+     42
    FP_OFF          20
    FP_SEG          20
    MK_FP           28
    addptr          16
    base            37      39      42
    farnormal       35
    num             16      23      26
    off             18      20      21+     23      25+     28
    p       16      20+
    para            37      40      41      42
    ptr             35      39      40
    seg             18      20      21      25      26      28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  FRACTION.C - Compute continued fraction series
**
**  cfrac() donated to the public domain by the author, Thad Smith
**  original Fraction.C, public domain by Bob Stout, modified to use cfrac()
*/

#include <math.h>
#include <float.h>

#define MAX_LENGTH 100

long double cfrac(long double x, long double *q, long double *q, int bits)
{
      double v;                                 /* integer in series    */
      long double del;                          /* x - v                */
      long double z;    /* approximated value from truncated series     */
      long double t;                            /* temp                 */
      long double p0 = 0.0, q0 = 0.0;           /* last p, q            */
      long double imax;                         /* max for p, q         */
      static long double cf[MAX_LENGTH]; /* continued fraction integers */
      int i, j, ntimes = MAX_LENGTH;;

      if (x < 0)
            x = -x;
      imax = floor(pow(2.0, bits)) - 1.0;
      for (i = 0; i < ntimes; i++)
      {
            v = floor((double)x);
            cf[i] = v;
            z = cf[i];
            *p = z; *q = 1;
            for (j = i; j--; )
            {
                  z = cf[j] + 1.0/z;
                  t = *p;
                  *p = cf[j] * (*p) + (*q);
                  *q = t;
            }
            del = x-v;
            if (del < DBL_EPSILON)
                  break;
            if ((*p > imax) || (*q > imax))
            {
                  *p = p0;
                  *q = q0;
                  break;
            }
            else
            {
                  p0 = *p;
                  q0 = *q;
            }
            x = 1.0 / del;
      }
      return (*p)/(*q);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main (int argc, char *argv[])
{
      long double x;      /* value to be approximated */
      long double r,p,q;  /* approx ratio r = p/q     */
      int bits;           /* bits of precision        */

      if (argc < 2 || argc > 3)
      {
            puts ("Use: FRACTION value [precision]");
            puts ("where value = floating point value to generate "
                  "continued fraction");
            puts ("      precision (optional) = bits in "
                  "numerator/denominator");
            return 1;
      }
      sscanf (argv[1], "%Lf", &x);
      if (argc == 3)
            bits = atoi(argv[2]);
      else  bits = 32;

      cfrac(x, &p, &q, bits);
      printf("\n[%.20Lf]\n%.0Lf/%.0Lf = %lXh/%lXh = %.20Lf\n",
            x, p, q, (long)p, (long)q, r = p/q);
      printf("Error = %.10Lg, (%.10Lf%%)\n", r - x, 100. * (r - x) / x);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

2829 characters
93 lines

LEXICAL STATISTICS

13 comments [std-C]
0 comments [C++]
7 preprocessor instructions
0 constants [character]
8 constants [string]
18 constants [numeric]

SYMBOL TABLE

DBL_EPSILON		43									
EXIT_SUCCESS		90									
MAX_LENGTH		13	23	24							
TEST	61										
argc	66	72+	82								
argv	66	81	83								
atoi	83										
bits	15	28	70	83	84	86					
cf	23	32	33	37	39						
cfrac	15	86									
del	18	42	43	56							
floor	28	31									
h	10	11	63	64							
i	24	29+	32	33	35						
imax	22	28	45+								
j	24	35+	37	39							
main	66										
math	10										
ntimes	24	29									
p	15	34	38	39+	45	47	53	58	69	86	88+
p0	21	47	53								
pow	28										
printf	87	89									
puts	74	75	77								
q	15	34	39	40	45	48	54	58	69	86	88+
q0	21	48	54								
r	69	88	89+								
sscanf	81										
stdio	63										
stdlib	64										
t	20	38	40								
v	17	31	32	42							
x	15	26	27+	31	42	56	68	81	86	88	89+
z	19	33	34	37+							

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  FRAND.C - Public domain by Larry Hudson
  5 | */
  6 |
  7 | #include <math.h>
  8 | #include <time.h>
  9 | #include "snipmath.h"
 10 |
 11 | #define TEN_PI 31.41592653589793
 12 | #define E      2.718281828459045
 13 |
 14 | /*-----------------------------------------------------------------+
 15 | |           Return random double between 0.0 and 1.0              |
 16 | |                                                                 |
 17 | |    If n is negative it will randomize the seed, based on the    |
 18 | |        current MSDOS time.                                      |
 19 | |    If n is zero it will return the next random number.          |
 20 | |    If n is positive it will set the seed to a value based on    |
 21 | |        the value of n.                                          |
 22 | +-----------------------------------------------------------------*/
 23 |
 24 | double frandom(int n)
 25 | {
 26 |       static double seed = E;
 27 |       double dummy;
 28 |       time_t tim;
 29 |
 30 |       if (n < 0)
 31 |       {
 32 |             time(&tim);
 33 |             seed = (double)tim;
 34 |       }
 35 |       else if (n > 0)
 36 |             seed = (double)n * E;
 37 |
 38 |       seed = modf(seed * TEN_PI + E, &dummy);
 39 |       return seed;
 40 | }
```

TEXT STATISTICS

```
  1160 characters
    40 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    E       12      26      36      38
    TEN_PI          11      38
    dummy           27      38
    frandom         24
    h        7       8
    math             7
    modf            38
    n       24      30      35      36
    seed            26      33      36      38+     39
    tim             28      32      33
    time             8      32
    time_t          28
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* fscanbin.c -- scan binary fields via format string
  4 | **
  5 | **  public domain by Ray Gardner   Englewood, Colorado   11/29/89
  6 | **
  7 | **  Usage: fscanbin(FILE *fp, char *format, ...)
  8 | **
  9 | **  where format string contains specifiers:
 10 | **   -ddd  means skip ddd bytes
 11 | **   i     means read a 16-bit int
 12 | **   l     means read a 32-bit int
 13 | **   sddd  means read a character string of up to ddd bytes
 14 | **           reads up to a nul byte if ddd is zero or missing
 15 | **   cnnn  means read a character field of nnn bytes (not nul-terminated)
 16 | **           reads one byte if nnn is zero or missing
 17 | */
 18 |
 19 | #include <stdlib.h>
 20 | #include <stdarg.h>
 21 | #include <ctype.h>
 22 | #include "fscanbin.h"
 23 |
 24 | #define SWAP16 0
 25 | #define SWAP32 0
 26 |
 27 | int fscanbin (FILE *fp, char *format, ...)
 28 | {
 29 |       va_list argp;
 30 |       unsigned char *p;
 31 |       unsigned k;
 32 |       int c;
 33 |       char *charp;
 34 |       WORD *WORDp;
 35 |       DWORD *DWORDp;
 36 |       int bytes_read;
 37 |
 38 |       bytes_read = 0;
 39 |       va_start(argp, format);
 40 |       for ( p = (unsigned char *)format; *p; )
 41 |       {
 42 |             switch( *p & 0xFF )
 43 |             {
 44 |             case '-':
 45 |                   for ( k = 0, c = *++p; isdigit(c); c = *++p )
 46 |                         k = 10 * k + c - '0';
 47 |                   if ( k == 0 )
 48 |                         k = 1;
 49 |                   if ( fseek(fp, (long)k, SEEK_CUR) )
 50 |                         return -2;     /* i/o error */
 51 |                   bytes_read += k;
 52 |                   break;
 53 |
 54 |             case 'i':
 55 |                   WORDp = va_arg(argp, WORD *);
 56 |                   if ( fread((void *)WORDp, sizeof(WORD), 1, fp) != 1 )
 57 |                         return -2;     /* i/o error */
 58 | #if SWAP16
 59 |                   WORDswap(WORDp);
 60 | #endif
 61 |                   p++;
 62 |                   bytes_read += sizeof(WORD);
 63 |                   break;
 64 |
 65 |             case 'l':
 66 |                   DWORDp = va_arg(argp, DWORD *);
 67 |                   if ( fread((void *)DWORDp, sizeof(DWORD), 1, fp) != 1 )
 68 |                         return -2;     /* i/o error */
 69 | #if SWAP32
 70 |                   DWORDswap(DWORDp);
 71 | #endif
 72 |                   p++;
 73 |                   bytes_read += sizeof(DWORD);
 74 |                   break;
 75 |
 76 |             case 's':
 77 |                   charp = va_arg(argp, char *);
 78 |                   for ( k = 0, c = *++p; isdigit(c); c = *++p )
 79 |                         k = 10 * k + c - '0';
 80 |                   do
 81 |                   {
 82 |                         c = getc(fp);
 83 |                         if ( c == EOF )
 84 |                               return -2;
 85 |                         *charp++ = (char)c;
 86 |                         bytes_read++;
 87 |                   } while ( c && (k == 0 || --k) );
 88 |                   break;
 89 |
 90 |             case 'c':
 91 |                   charp = va_arg(argp, char *);
 92 |                   for ( k = 0, c = *++p; isdigit(c); c = *++p )
 93 |                         k = 10 * k + c - '0';
 94 |                   if ( k == 0 )
 95 |                         k = 1;
 96 |                   if ( fread((void *)charp, sizeof(char), k, fp) != k )
 97 |                         return -2;     /* i/o error */
 98 |                   bytes_read += k;
 99 |                   break;
100 |
```

```
 101 |             default:
 102 |                   return -1; /* bad format */
 103 |             }
 104 |       }
 105 |       va_end(argp);
 106 |       return bytes_read;
 107 | }
```

TEXT STATISTICS

```
  3329 characters
   107 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     8 constants [character]
     1 constants [string]
    25 constants [numeric]
```

SYMBOL TABLE

```
    DWORD           35      66      67      73
    DWORDp          35      66      67      70
    DWORDswap       70
    EOF             83
    FILE            27
    SEEK_CUR        49
    SWAP16          24      58
    SWAP32          25      69
    WORD            34      55      56      62
    WORDp           34      55      56      59
    WORDswap        59
    argp            29      39      55      66      77      91     105
    bytes_read              36      38      51      62      73      86      98     106
    c       32      45+     46      78+     79      82      83      85      87      92+     93
    charp           33      77      85      91      96
    ctype           21
    format          27      39      40
    fp              27      49      56      67      82      96
    fread           56      67      96
    fscanbin        27
    fseek           49
    getc            82
    h       19      20      21
    isdigit         45      78      92
    k       31      45      46+     47      48      49      51      78      79+     87+     92      93+
            94      95      96+     98
    p       30      40+     42      45+     61      72      78+     92+
    stdarg          20
    stdlib          19
    va_arg          55      66      77      91
    va_end         105
    va_list         29
    va_start        39
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPPETS header file for FSCANBIN.C
  5 | */
  6 |
  7 | #ifndef FSCANBIN__H
  8 | #define FSCANBIN__H
  9 |
 10 | #include <stdio.h>
 11 | #include "sniptype.h"
 12 |
 13 | #define WORDswap(n)  (*n = (*n << 8) | (*n >> 8))
 14 |
 15 | #define DWORDswap(n)  (\
 16 |                         WORDswap(&((WORD *)n)[0]),\
 17 |                         WORDswap(&((WORD *)n)[1]),\
 18 |                         *n = (*n << 16) | (*n >> 16)\
 19 |                         )
 20 |
 21 | #define maxk 32767
 22 |
 23 | int fscanbin (FILE *fp, char *format, ...);
 24 |
 25 | #endif /* FSCANBIN__H */
```

TEXT STATISTICS

```
   548 characters
    25 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     1 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    DWORDswap       15
    FILE            23
    FSCANBIN__H              7       8
    WORD            16      17
    WORDswap        13      16      17
    format          23
    fp              23
    fscanbin        23
    h       10
    maxk            21
    n       13+     15      16      17      18+
    stdio           10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Portable PC screen functions
**  Public domain by Bob Stout
**  Uses SCRNMACS.H and HUGEREAD.C, also from SNIPPETS
*/

#include <stdio.h>
#include "sniptype.h"
#include "scrnmacs.h"

/*
**  Prototypes from HUGEREAD.C, also in SNIPPETS
*/

long hugefread(FILE *fp, char FAR *buf, long size);
long hugefwrite(FILE *fp, char FAR *buf, long size);

/*
**  Save the text screen to a file
*/

Boolean_T fSaveScrn(const char *fname)
{
      FILE *file;

      if (NULL == (file = fopen(fname, "wb")))
            return Error_;
      if ((long)SCRNBYTES != hugefwrite(file, (char FAR *)SCRBUFF,
            (long)SCRNBYTES))
      {
            return Error_;
      }
      fclose(file);
      return Success_;
}

/*
**  Restore the text screen from a file
*/

Boolean_T fRestoreScrn(const char *fname)
{
      FILE *file;

      if (NULL == (file = fopen(fname, "rb")))
            return Error_;
      if ((long)SCRNBYTES != hugefread(file, (char FAR *)SCRBUFF,
            (long)SCRNBYTES))
      {
            return Error_;
      }
      fclose(file);
      return Success_;
}

#ifdef TEST

#include <conio.h>

/*
**  Run this test with a screenful of misc. stuff
**
**  Note that this test requires that VIDPORT.C and SCROLL.C, also from
**  SNIPPETS, be linked.
*/

main()
{
      if (Error_ == fSaveScrn("fscrnsav.tst"))
      {
            puts("Unable to save the screen");
            return 1;
      }
      ClrScrn(7);
      GotoXY(0, 0);
      fputs("fClrScrn() tested", stderr);
      fputs("\nHit any key to continue...\n", stderr);
      getch();
      fRestoreScrn("fscrnsav.tst");
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1765 characters
    85 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     9 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    Boolean_T       24      43
    ClrScrn         76
    Error_          29      33      48      52      71
    FAR             17      18      30      49
    FILE            17      18      26      45
    GotoXY          77
    NULL            28      47
    SCRBUFF         30      49
    SCRNBYTES       30      31      49      50
    Success_        36      55
    TEST            58
    buf             17      18
    conio           60
    fRestoreScrn            43      81
    fSaveScrn       24      71
    fclose          35      54
    file            26      28      30      35      45      47      49      54
    fname           24      28      43      47
    fopen           28      47
    fp              17      18
    fputs           78      79
    getch           80
    h       9       60
    hugefread       17      49
    hugefwrite              18      30
    main            69
    puts            73
    size            17      18
    stderr          78      79
    stdio            9
```

```
/* +++Date last modified: 05-Jul-1997 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include "bacstd.h"

MODULEINF("1992-08-01", "1990-1992 Erik Bachmann (E-mail: ebp@dde.dk" ) ;

#define  BUFFERSIZE  512

/*
 /=======================================\
|     FIND_STR_IN_FILE                    |-----------------------------------|
|\=======================================/
|
| Searches a binary file for a string.
|
| Returns the start offset for the first occurrence of the string.
|
|
|
|
|-------------------------------------------------------------------------------|
| CALL:
|    find_str_in_file(Filename, startoffset, string);
|
| HEADERS:
|    stdio.h
|    string.h
|    malloc.h
|    io.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    fpFile      : Pointer to the (open) binary file
|    lOffset     : Startoffset for search
|    pszStr      : String to search for
|
| PROTOTYPE:
|    long _CfnTYPE find_str_in_file(FILE *fpFile, long lOffset,
|                                   char *pszStr);
|
| RETURN VALUE:
|    long lStatusFlag   : -1   : String not found
|                         -2   : Not enough memory to perform search
|                        else : Offset for string in file
|
| MODULE:
|    fsif.c
|-------------------------------------------------------------------------------|
|1994-03-18/Bac
|   -  Enhanced error detection. Returning error value
|1995-06-19/Bac
|   -  Patch for returning position. Changed from blockwise to bytewise
|      calculation.
|1995-09-20/Bac
|  -  Ajusted secondary search in last block
|
|1995-10-27/Bac
|  -  Released to public domain
|
|-------------------------------------------------------------------------------|
|1992-08-01/Erik Bachmann
\===============================================================================|*/

long _CfnTYPE find_str_in_file(FILE *fpFile, long lOffset, char *pszStr)
{
      long  lStatusFlag = -1l,            /* Status: -1 = not found     */
            lPosInFile  = 0l,             /* Position in file           */
            lCurrentPos = 0l;

      char  cFound      = FALSE,          /* Flag: FALSE = Not found (yet) */
            *pszBuffer  = NULL;           /* Buffer for fileinput       */

      int   iDataSize   = 0,              /* Size of data read from file*/
            iSector     = 0,              /* No of blocks read          */
            offset      = 0,              /* local counter              */
            iBufferSize = BUFFERSIZE,
            iNoBlocks   = 0;              /* No of blocks remaining     */

      /*-------------------------------------------------------------*/

      lPosInFile = filelength(fileno(fpFile));  /* Find filesize        */

      iNoBlocks = (int)(lPosInFile / (int) iBufferSize) ;
                                    /* Calculate remaining no of blocks */

      fseek(fpFile, lOffset, SEEK_SET) ;  /* Go to start offset */

      lCurrentPos = lOffset ;

      iNoBlocks = (int)((lPosInFile - lOffset) / (int) iBufferSize);
                                    /* Calculate remaining no of blocks */

      if (0 == (pszBuffer = (char *) calloc(2 * iBufferSize, sizeof(char))))
            return -2;              /* Allocate buffer                  */
```

```
101 |
102 |       memset(pszBuffer, '\0', iBufferSize);
103 |
104 |       iDataSize = fread(pszBuffer, sizeof(char), iBufferSize, fpFile);
105 |                                   /* Read the first block             */
106 |
107 |       while ((0 < iNoBlocks) && !cFound)
108 |       {                           /* Repeat until EOF or found        */
109 |             iSector++;            /* Counting no of blocks read       */
110 |             iNoBlocks--;
111 |
112 |             memset(&pszBuffer[iBufferSize], '\0', iBufferSize);
113 |             iDataSize = fread(&pszBuffer[iBufferSize], sizeof(char),
114 |                               iBufferSize, fpFile); /* Read next block  */
115 |
116 |             for (offset = 0; offset < iBufferSize; offset++)
117 |                                   /* Search first block               */
118 |             {
119 |                   if (0 == strncmp(&pszBuffer[ offset ],
120 |                                    pszStr, strlen(pszStr)))
121 |                   {               /* Is the string placed here?       */
122 |                         cFound = TRUE;    /* Yes -> set flag          */
123 |                         break;
124 |                   }
125 |                   else  lCurrentPos++;    /* No -> Try again          */
126 |             }
127 |
128 |             memcpy(pszBuffer, &pszBuffer[ iBufferSize ], iBufferSize);
129 |                                   /* Shift block left                 */
130 |       }
131 |
132 |       /* Search the last Sector read if tag not found yet              */
133 |
134 |       if (!cFound)
135 |       {
136 |             iSector++;            /* Counting no of blocks read       */
137 |
138 |             for (offset = 0; offset < iDataSize - strlen(pszStr); offset++)
139 |                   if (0 == strncmp(&pszBuffer[ offset ], pszStr,
140 |                                    strlen(pszStr)))
141 |                   {               /* Is the string placed in last block?*/
142 |                         cFound = TRUE;    /* Found -> set flag        */
143 |                         break;
144 |                   }
145 |                   else if (lCurrentPos < lPosInFile)
146 |                                   /* Check for End of File            */
147 |                         lCurrentPos++;    /* In file -> goto next     */
148 |                   else  break;            /* End of File = quit       */
149 |       }
150 |
151 |       free(pszBuffer);            /* Free the allocated memory again  */
152 |
153 |       if (TRUE == cFound)               /* IF tag is found            */
154 |             return ((long) lCurrentPos);  /* Return bytewise positon  */
155 |       else  return lStatusFlag;         /* Return errorcode           */
156 | }
```

TEXT STATISTICS

```
  5905 characters
   156 lines
```

LEXICAL STATISTICS

```
    36 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     2 constants [character]
     3 constants [string]
    16 constants [numeric]
```

SYMBOL TABLE

```
    BUFFERSIZE              11      82
    FALSE           76
    FILE            70
    MODULEINF        9
    NULL            77
    SEEK_SET        92
    TRUE           122     142     153
    _CfnTYPE        70
    cFound          76     107     122     134     142     153
    calloc          99
    filelength              87
    fileno          87
    find_str_in_file        70
    fpFile          70      87      92     104     114
    fread          104     113
    free           151
    fseek           92
    h        3       4       5       6
    iBufferSize             82      89      96      99     102     104     112+    113     114     116
           128+
    iDataSize       79     104     113     138
    iNoBlocks       83      89      96     107     110
    iSector         80     109     136
    io               6
    lCurrentPos             74      94     125     145     147     154
    lOffset         70      92      94      96
    lPosInFile              73      87      89      96     145
    lStatusFlag             72     155
    memcpy         128
    memset         102     112
    offset          81     116+    119     138+    139
    pszBuffer       77      99     102     104     112     113     119     128+    139     151
    pszStr          70     120+    138     139     140
    stdio            3
    stdlib           5
    string           4
    strlen         120     138     140
    strncmp        119     139
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ======================================================================
  4 |     FSM.c           Finite State machines.
  5 |                     Version 0.22x, 93-08-05.
  6 |
  7 |  _____              This version is Public Domain.
  8 |  /_|__|             A.Reitsma, Delft, Nederland.
  9 | /  | \  ------------------------------------------------------------ */
 10 |
 11 | #include <stdlib.h>
 12 | #include "ll_defs.h"
 13 | #include "fsm.h"
 14 |
 15 | /* tmp definitions until error module is finished */
 16 | #define MEMORY      -1
 17 | #define Error(x)    return x
 18 |
 19 | /*  --- Local definitions and data ---------------------------------- */
 20 | struct state_info
 21 | {
 22 |     struct FSMstate_entry * table ;
 23 |     int entries ;
 24 | };
 25 |
 26 | static int state ;      /* the machine's state or it's maximum in setup */
 27 | static struct state_info * info ;  /* derived info in a handier format  */
 28 |
 29 | /*  --- Setup and action functions ----------------------------------- */
 30 |
 31 | int FSMsetup( struct FSMstate_entry * StateEntry )
 32 | {
 33 |     struct FSMstate_entry * Entry = StateEntry ;
 34 |     struct state_info * data ;
 35 |
 36 |     /*  Find 'highest' state number. Negative number is end of list.
 37 |         Gaps allowed. Must be sorted on state number in current version.
 38 |         Prepared for non-sorted version though.
 39 |     */
 40 |     state = 0 ;
 41 |     do
 42 |     {
 43 |         if( state < Entry->state )
 44 |             state = Entry->state ;
 45 |
 46 |     }while( (++Entry)->state >= 0 );
 47 |
 48 |     /*  Create info 'array'
 49 |     */
 50 |     if( NULL == (info = MALLOC( state+1, struct state_info )))
 51 |         Error( MEMORY );
 52 |
 53 |     /*  Setup info 'array': split into sub-lists with size info.
 54 |         First entry:
 55 |     */
 56 |     data = info ;
 57 |     data->table = StateEntry ;
 58 |     data->entries = 1 ;
 59 |     /*  ... and the second and following entries:
 60 |     */
 61 |     while( (++StateEntry)->state >= 0 )
 62 |     {
 63 |         if( data->table->state == StateEntry->state )
 64 |             data->entries ++ ;          /* same state */
 65 |         else
 66 |         {                               /* new state  */
 67 |             data = info + StateEntry->state ;
 68 |             data->table = StateEntry ;
 69 |             data->entries = 1 ;
 70 |         }
 71 |     }
 72 |
 73 |     state = 0 ; /* initial state for state machine */
 74 |
 75 |     return 0 ;
 76 | }
 77 |
 78 | int FSMaction( int Cond )
 79 | {
 80 |     int count ;
 81 |     struct FSMstate_entry * Table = info[ state ].table ;
 82 |
 83 |     /*  Find matching condition in second or higher entry in table.
 84 |     */
 85 |     for( count = 1 ; count < info[ state ].entries ; count ++ )
 86 |     {
 87 |         Table ++;
 88 |         if( Cond == Table->cond )
 89 |         {
 90 |             state = Table->next ;      /* match found: adjust state,    */
 91 |                                        /* and execute non-NULL funtion. */
 92 |             if( NULL != Table->action )
 93 |                 return Table->action( Cond );
 94 |             else
 95 |                 return 0 ;
 96 |         }
 97 |     }
 98 |
 99 |     /*  No matching condition found: use first entry as default action.
100 |     */
```

```
 101 |     Table = info[ state ].table ;
 102 |     state = Table->next ;
 103 |     return Table->action( Cond );
 104 | }
 105 |
 106 | /* === FSM.c ============================================================ */
```

TEXT STATISTICS

```
  3141 characters
   106 lines
```

LEXICAL STATISTICS

```
    19 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    Cond            78      88      93     103
    Entry           33      43      44      46
    Error           17      51
    FSMaction       78
    FSMsetup        31
    FSMstate_entry          22      31      33      81
    MALLOC          50
    MEMORY          16      51
    NULL            50      92
    StateEntry              31      33      57      61      63      67      68
    Table           81      87      88      90      92      93     101     102     103
    action          92      93     103
    cond            88
    count           80      85+
    data            34      56      57      58      63      64      67      68      69
    entries         23      58      64      69      85
    h       11
    info            27      50      56      67      81      85     101
    next            90     102
    state           26      40      43+     44+     46      50      61      63+     67      73      81
            85      90     101     102
    state_info              20      27      34      50
    stdlib          11
    table           22      57      63      68      81     101
    x       17+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ======================================================================
  4 |     FSM.h           Finite State machines.
  5 |                     Version 0.01x, 93-08-04.
  6 |
  7 |  _____              This version is Public Domain.
  8 |  /_|__|             A.Reitsma, Delft, Nederland.
  9 | /  | \  ------------------------------------------------------------ */
 10 |
 11 | struct FSMstate_entry
 12 | {
 13 |     int state ;
 14 |     int cond  ;
 15 |     int next  ;
 16 |     int (*action)(int);
 17 | };
 18 |
 19 | #define STATE_LIST_END  { -1, -1, 0, NULL }
 20 | #define LIST_DEFAULT   -1
 21 |
 22 | int FSMsetup( struct FSMstate_entry * StateTable );
 23 | int FSMaction( int Cond );
 24 |
 25 | /* === FSM.h ============================================================ */
```

TEXT STATISTICS

```
   733 characters
    25 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     0 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    Cond            23
    FSMaction       23
    FSMsetup        22
    FSMstate_entry          11      22
    LIST_DEFAULT            20
    NULL            19
    STATE_LIST_END          19
    StateTable              22
    action          16
    cond            14
    next            15
    state           13
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  code snippet demonstrating a finite state machine (FSM)
*/

typedef enum {s0,s1,s2,s3,s4,...,sn,sexit} state;

state nextstate;
int done = 0;

nextstate = s0;  /* set up to start with the first state */
while(!done)
   switch(nextstate)
      {
         case s0:
            nextstate = do_state_0();
            break;
         case s1:
            nextstate = do_state_1();
            break;
         case s2:
            nextstate = do_state_2();
            break;
         case s3:
             .
             .
             .
         case sn:
            nextstate = do_state_n();
            break;
         case sexit:
            done = TRUE;
            break;
         default:
            /*  some sort of unknown state */
            break;
      }
```

TEXT STATISTICS

```
   867 characters
    39 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    TRUE            34
    do_state_0              17
    do_state_1              20
    do_state_2              23
    do_state_n              31
    done            10      13      34
    nextstate        9      12      14      17      20      23      31
    s0               7      12      16
    s1               7      19
    s2               7      22
    s3               7      25
    s4               7
    sexit            7      33
    sn               7      30
    state            7       9
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Public domain by Jeff Dunlop & Bob Stout
*/

#include <time.h>
#include "ftime.h"

#if !defined(__TURBOC__) && !defined(__SC__) && !defined(__POWERC)

#ifdef __ZTC__
 #pragma ZTC align 1
 #define DOS_GETFTIME dos_getftime
 #define DOS_SETFTIME dos_setftime
 typedef unsigned FTIME_T_;
#else
 #pragma pack(1)
 #define DOS_GETFTIME _dos_getftime
 #define DOS_SETFTIME _dos_setftime
 #ifdef __WATCOMC__
  typedef unsigned short FTIME_T_;
 #else
  typedef unsigned FTIME_T_;
 #endif
#endif

int getftime (int handle, struct ftime *ftimep)
{
      int retval = 0;
      union
      {
            struct
            {
                  unsigned time;
                  unsigned date;
            } msc_time;
            struct ftime bc_time;
      } FTIME;

      if (0 == (retval = DOS_GETFTIME(handle,
            (FTIME_T_ *)&FTIME.msc_time.date,
            (FTIME_T_ *)&FTIME.msc_time.time)))
      {
            *ftimep = FTIME.bc_time;
      }
      return retval;
}

int setftime (int handle, struct ftime *ftimep)
{
      union
      {
            struct
            {
                  unsigned time;
                  unsigned date;
            } msc_time;
            struct ftime bc_time;
      } FTIME;

      FTIME.bc_time = *ftimep;

      return DOS_SETFTIME(handle, FTIME.msc_time.date, FTIME.msc_time.time);
}

#endif

static void _ftimecnvrt(struct ftime *ft, struct tm *time, time_t *tt)
{
      time->tm_sec  = ft->ft_tsec * 2;
      time->tm_min  = ft->ft_min;
      time->tm_hour = ft->ft_hour;
      time->tm_mday = ft->ft_day;
      time->tm_mon  = ft->ft_month - 1;
      time->tm_year = ft->ft_year + 80;
      *tt = mktime(time);                 /* Fill in rest of the struct */
}

void ftime2tm(struct ftime *ft, struct tm *time)
{
      time_t tt;

      _ftimecnvrt(ft, time, &tt);
}

time_t ftime2time(struct ftime *ft)
{
      struct tm time;
      time_t tt;

      _ftimecnvrt(ft, &time, &tt);
      return tt;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>
#if defined(MSDOS) || defined(__MSDOS__)
```

```
101 |  #include "unistd.h"
102 | #else
103 |  #include <unistd.h>
104 | #endif
105 | #include "errors.h"
106 | #include "sniptype.h"
107 |
108 | #ifdef __WATCOMC__
109 |  #pragma off (unreferenced);
110 | #endif
111 | #ifdef __TURBOC__
112 |  #pragma argsused
113 | #endif
114 |
115 | main(int argc, char *argv[])
116 | {
117 |       struct ftime ft;
118 |       struct tm time;
119 |       FILE *fp;
120 |       char timeline[80];
121 |
122 |       fp = cant(argv[0], "r");
123 |       if (Success_ != getftime(fileno(fp), &ft))
124 |             ErrExit("getftime() failed");
125 |       ftime2tm(&ft, &time);
126 |       printf("ft_tsec = %d\n", ft.ft_tsec);
127 |       printf("ft_min  = %d\n", ft.ft_min);
128 |       printf("ft_hour = %d\n", ft.ft_hour);
129 |       printf("ft_day  = %d\n", ft.ft_day);
130 |       printf("ft_month= %d\n", ft.ft_month);
131 |       printf("ft_year = %d\n", ft.ft_year);
132 |       puts("");
133 |       printf("tm_sec  = %d\n", time.tm_sec);
134 |       printf("tm_min  = %d\n", time.tm_min);
135 |       printf("tm_hour = %d\n", time.tm_hour);
136 |       printf("tm_mday = %d\n", time.tm_mday);
137 |       printf("tm_mon  = %d\n", time.tm_mon);
138 |       printf("tm_year = %d\n", time.tm_year);
139 |
140 |       strftime(timeline, 80, "\n%c\n", &time);
141 |       puts(timeline);
142 |       return EXIT_SUCCESS;
143 | }
144 |
145 | #endif /* TEST */
```

TEXT STATISTICS

```
  3378 characters
   145 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    33 preprocessor instructions
     0 constants [character]
    20 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    DOS_GETFTIME            14      19      41
    DOS_SETFTIME            15      20      64
    EXIT_SUCCESS           142
    ErrExit        124
    FILE           119
    FTIME           39      42      43      45      60      62      64+
    FTIME_T_        16      22      24      42      43
    MSDOS          100
    Success_       123
    TEST            96
    ZTC             13
    __MSDOS__      100
    __POWERC        10
    __SC__          10
    __TURBOC__              10     111
    __WATCOMC__             21     108
    __ZTC__         12
    _dos_getftime           19
    _dos_setftime           20
    _ftimecnvrt             69      84      92
    align           13
    argc           115
    argsused       112
    argv           115     122
    bc_time         38      45      59      62
    cant           122
    date            36      42      57      64
    defined         10+    100+
    dos_getftime            14
    dos_setftime            15
    fileno         123
    fp             119     122     123
    ft              69      71      72      73      74      75      76      80      84      87      92
           117     123     125     126     127     128     129     130     131
    ft_day          74     129
    ft_hour         73     128
    ft_min          72     127
    ft_month        75     130
    ft_tsec         71     126
    ft_year         76     131
    ftime           28      38      50      59      69      80      87     117
    ftime2time              87
    ftime2tm        80     125
    ftimep          28      45      50      62
    getftime        28     123
    h        7      98      99     103
    handle          28      41      50      64
    main           115
    mktime          77
    msc_time        37      42      43      58      64+
    off            109
    pack            18
    printf         126     127     128     129     130     131     133     134     135     136     137
           138
    puts           132     141
    retval          30      41      47
    setftime        50
    stdio           98
    stdlib          99
    strftime       140
    time             7      35      43      56      64      69      71      72      73      74      75
            76      77      80      84      89      92     118     125     133     134     135     136
           137     138     140
    time_t          69      82      87      90
    timeline       120     140     141
    tm              69      80      89     118
    tm_hour         73     135
    tm_mday         74     136
    tm_min          72     134
    tm_mon          75     137
    tm_sec          71     133
    tm_year         76     138
    tt              69      77      82      84      90      92      93
    unistd         103
    unreferenced           109
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Public domain by Jeff Dunlop
  5 | */
  6 |
  7 | #ifndef FTIME__H
  8 | #define FTIME__H
  9 |
 10 | #include <dos.h>
 11 | #include <time.h>
 12 |
 13 | #if !defined(__TURBOC__) && !defined(__SC__) && !defined(__POWERC)
 14 |
 15 | struct ftime /* As defined by Borland & Symantec */
 16 | {
 17 |     unsigned    ft_tsec  : 5;   /* Two second interval */
 18 |     unsigned    ft_min   : 6;   /* Minutes */
 19 |     unsigned    ft_hour  : 5;   /* Hours */
 20 |     unsigned    ft_day   : 5;   /* Days */
 21 |     unsigned    ft_month : 4;   /* Months */
 22 |     unsigned    ft_year  : 7;   /* Year */
 23 | };
 24 |
 25 | int getftime (int, struct ftime *);
 26 | int setftime (int, struct ftime *);
 27 |
 28 | #else
 29 |  #include <io.h>
 30 | #endif
 31 |
 32 | void   ftime2tm(struct ftime *ft, struct tm *time);
 33 | time_t ftime2time(struct ftime *ft);
 34 |
 35 | #endif /* FTIME__H */
```

TEXT STATISTICS

```
   809 characters
    35 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     0 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    FTIME__H         7       8
    __POWERC        13
    __SC__          13
    __TURBOC__              13
    defined         13+
    dos             10
    ft              32      33
    ft_day          20
    ft_hour         19
    ft_min          18
    ft_month        21
    ft_tsec         17
    ft_year         22
    ftime           15      25      26      32      33
    ftime2time              33
    ftime2tm        32
    getftime        25
    h       10      11      29
    io              29
    setftime        26
    time            11      32
    time_t          33
    tm              32
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************/
/*                                                                           */
/*  FUPDATE.C                                                                */
/*                                                                           */
/*  Functions to flush  disk buffers to disk.                                */
/*                                                                           */
/*  Original Copyright 1990-93 by Robert B. Stout as part of                 */
/*  the MicroFirm Function Library (MFL)                                     */
/*                                                                           */
/*  The user is granted a free limited license to use this source file       */
/*  to create royalty-free programs, subject to the terms of the             */
/*  license restrictions specified in the LICENSE.MFL file.                  */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <io.h>
#include "dosfiles.h"

/*
**  fdupdate()
**
**  Update a file by handle reference.
**
**  Arguments: 1 - Handle of file to update
**
**  Returns: Success_ or error code
*/

int fdupdate(int fd)
{
      if (3 > _osmajor || (3 == _osmajor && 3 > _osminor))
      {
            return close(dup(fd));
      }
      else
      {
            union REGS regs;

            regs.h.ah = 0x68;
            regs.x.bx = fd;
            intdos(&regs, &regs);
            if (regs.x.cflag)
                  return regs.x.ax;
            else  return Success_;
      }
}

/*
**  fupdate()
**
**  Update a C buffered file.
**
**  Arguments: 1 - FILE pointer of file to update
**
**  Returns: Success_ or error code
*/

int fupdate(FILE *fp)
{
      fflush(fp);
      return fdupdate(fileno(fp));
}
```

```
TEXT STATISTICS

  2016 characters
    67 lines

LEXICAL STATISTICS

    17 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    FILE            63
    REGS            42
    Success_        49
    _osmajor        36+
    _osminor        36
    ah              44
    ax              48
    bx              45
    cflag           47
    close           38
    dos             20
    dup             38
    fd              34      38      45
    fdupdate        34      66
    fflush          65
    fileno          66
    fp              63      65      66
    fupdate         63
    h       18      19      20      21      44
    intdos          46
    io              21
    regs            42      44      45      46+     47      48
    stdio           18
    stdlib          19
    x       45      47      48
```

```
/* +++Date last modified: 05-Jul-1997 */

/* getcmt.c - get comments from a C or C++ source file    */
/*
                                Byte_Magic Software
                              9850 Meadowglen Ln. #35
                                Houston, Tx.  77042
                                  (713) 975-9033

Author:     Greg Messer
Program:    getcmt.c
Purpose:    Isolate comments from a C or C++ source file. Filter.
Syntax:     getcmt [/L?] [filename]
            (may use redirection for file and/or device I/O)
Release:    Released to the Public Domain by Byte_Magic Software.
Date:       07-11-88 GM...
                First release.
Revised:    01-13-90 GM...
                Streamlined logic.
            01-15-90 Bob Stout (RBS)...
                Fixed unsigned char error as return from getc().
            01-22-90 GM...
                Fixed bug handling "xx/" (xx = **) at end of comment.
                Added C++ comment extraction.
                Added return of count to OS (ERRORLEVEL in MS-DOS).
            01-24-90 RBS
                Added filename spec option
                Added /? switch
System:     Compiled with Zortech C V2.06 under MS-DOS 3.20
            for IBM PCs and compatibles.
Rules:      ANSI C comments begin with /x and end with x/. (x = *).
            Comments do not nest and do not appear in string or character
            constants.
            C++ comments begin with double slashes and end at EOL.
            A Microsoft extension to C allows C++ style comments to serve as
            single-line comments in C source.
Comments:   Useful for creating documentation and improving commenting style.
            Input may be from a specified filename or stdin.
            Output is to stdout, so use DOS redirection for output.
            Messages go to stderr so they are not redirectable from the screen.
            Returns ERRORLEVEL = number of comments in source file(s).
Examples:
            Example... Output to screen:
            getcmt < cfile.c
               (displays comments from cfile.c on screen)
            type cfile.c | getcmt
               (same as above but slightly slower)
            getcmt < cfile.c | more
               (same as above, but pauses after each full screen)
            getcmt cfile.c /l | more
               (same as above, but display line numbers)

            Example... Output to printer:
            getcmt < cfile.c > prn
               (same as above but prints output on printer)
            type cfile.c | getcmt > prn
               (same as above but slightly slower)

            Example... Output to file:
            getcmt < cfile.c > cfile.cmt
               (writes cfile.c comments to cfile.cmt, overwriting existing file)
            getcmt < cfile.c >> cfile.doc
               (writes cfile.c comments to end of cfile.doc (appends))

            getcmt /?
               (displays help screen, returns ERRORLEVEL = 0)
            getcmt /x
               (invalid option - displays help screen, returns ERRORLEVEL = -1)

            For complete instructions on using redirection symbols, consult
            the PC-DOS or MS-DOS manual or a general DOS reference book.
*/

#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define LOOKS_GREAT 1
#define LESS_FILLING 0

int extract_c_cmts(void);
void inside_c_cmt(int);

FILE *infile = stdin;                   /* read input from here             */
int  show_nos = 0;                      /* 0 = don't display line numbers   */
int  line_no = 1;                       /* line_no = line number            */

/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */

main(int argc, char *argv[])                            /* main logic:      */
{
    register int i;
    const char *hype =
        "\nGETCMT v1.1 - GET CoMMenTs\nby Byte_Magic Software\n";
    const char *help =
        "\nUsage: GETCMT [/l?] [filename | <sourcefile.ext] "
        "[>destination file or device]\n"
        "Options:  l - Print line numbers\n"
        "          ? - Help\n"
```

```
101 |         "\nFilename optional - Reads source code from stdin "
102 |         "(Ctrl-C to quit before EOF)\n";
103 |     const char *oops =
104 |         "\a*** GETCMT - Can't open input file ";
105 |                                         /* display messages to operator */
106 | #if LOOKS_GREAT
107 |     fputs(hype, stderr);
108 | #elif LESS_FILLING
109 |     i = 0;
110 |     while(hype[i] != '\0')
111 |         putc(hype[i++], stderr);
112 | #endif
113 |
114 |     if (1 < argc)
115 |     {
116 |         for (i = 1; i < argc; ++i)
117 |         {
118 |             if ('/' == *argv[i])
119 |             {
120 |                 if ('l' == tolower(argv[i][1]))
121 |                     show_nos = 1;
122 |                 else
123 |                 {
124 |                     int ercode;
125 |
126 |                     ercode = ('?' == argv[i][1]) ? 0 : -1;
127 | #if LOOKS_GREAT
128 |                     fputs(help, stderr);
129 | #elif LESS_FILLING
130 |                     i = 0;
131 |                     while(help[i] != '\0')
132 |                         putc(help[i++], stderr);
133 | #endif
134 |                     if (ercode)             /* output BEL if invalid switch */
135 |                         putc('\a', stderr);
136 |                     return(ercode);
137 |                 }
138 |             }
139 |             else
140 |             {
141 |                 infile = fopen(argv[i], "r");
142 |                 if (!infile)
143 |                 {
144 | #if LOOKS_GREAT
145 |                     fputs(oops, stderr);
146 |                     fputs(argv[i], stderr);
147 | #elif LESS_FILLING
148 |                     char *p = argv[i];
149 |
150 |                     i = 0;
151 |                     while (oops[i])
152 |                         putc(oops[i], stderr);
153 |                     i = 0;
154 |                     while (*p)
155 |                         putc(*p++, stderr);
156 | #endif
157 |                 }
158 |             }
159 |         }
160 |     }
161 |
162 |     i = extract_c_cmts();                   /* extract comments in infile   */
163 |     putc('\n', stdout);
164 |
165 |     return(i);                              /* return number of comments to */
166 |                                             /* OS (ERRORLEVEL in DOS)       */
167 | }
168 |
169 | /*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
170 |
171 | int extract_c_cmts()                        /* comment extraction logic:    */
172 | {
173 |     register int chi, cht;              /* chi = char in, cht = char test   */
174 |     int count;                          /* count = comment count            */
175 |
176 |     count = 0;
177 |     chi = getc(infile);
178 |     while(chi != EOF)                       /* as long as there is input... */
179 |     {
180 |         if(chi == '/')                      /* process comments             */
181 |         {
182 |             cht = getc(infile);
183 |             if(cht == EOF)
184 |                 return(count);
185 |             if(cht == '*' || cht == '/')    /* if start of a comment...     */
186 |             {
187 |                 count++;                    /* count it and                 */
188 |                 inside_c_cmt(cht);          /* output all of the comment    */
189 |             }
190 |             else
191 |                 ungetc(cht, infile);
192 |         }
193 |         if ('\n' == chi)
194 |             line_no += 1;
195 |         chi = getc(infile);                 /* continue scanning input     */
196 |     }
197 |     return(count);
198 | }
199 |
200 | /*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
```

```
201 |
202 | char *lntoaz(void)                      /* line number to zero-padded ASCII */
203 | {
204 |     int i, num = line_no;
205 |     static char numbuf[] = "0000: ";
206 |
207 |     if (9999 < num)
208 |         strncpy(numbuf, "0000", 4);
209 |     else for (i = 3; i >= 0; --i)
210 |     {
211 |         numbuf[i] = (char)('0' + num % 10);
212 |         num /= 10;
213 |     }
214 |     return numbuf;
215 | }
216 |
217 | /*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
218 |
219 | void inside_c_cmt(int ch)                   /* comment output logic:        */
220 | {                                           /* input ch = either '*' for C  */
221 |                                             /* or '/' for C++               */
222 |
223 |     register int chi, cht;              /* chi = char in, cht = char test   */
224 | #if LESS_FILLING
225 |     char *p;
226 | #endif
227 |
228 |     if(ch == '/')                           /* make ch = '\n' if C++        */
229 |         ch = '\n';                          /* note: ch is already 1st char */
230 |                                             /* of end comment if this is C  */
231 |     putc('\n', stdout);
232 |     if (show_nos)
233 |     {
234 | #if LOOKS_GREAT
235 |     fputs(lntoaz(), stdout);
236 | #elif LESS_FILLING
237 |     p = lntoaz();
238 |         while (*p)
239 |         putc(*p++, stdout);
240 | #endif
241 |     }
242 |     chi = getc(infile);
243 |     while(chi != EOF)                       /* as long as there is input... */
244 |     {                                       /* process comments             */
245 |         if(chi == ch)
246 |         {
247 |             if(ch == '\n')                  /* if C++ comment is ended...   */
248 |                 return;                     /* stop outputting              */
249 |             cht = getc(infile);
250 |             if(cht == '/')                  /* if C comment is ended...     */
251 |                 return;                     /* stop outputting              */
252 |             else
253 |             {
254 |                 ungetc(cht, infile);
255 |                 putc(chi, stdout);
256 |             }
257 |         }
258 |         else
259 |             putc(chi, stdout);              /* else comment text, output it */
260 |         if ('\n' == chi)
261 |             line_no += 1;
262 |
263 |         chi = getc(infile);                  /* continue scanning input     */
264 |     }
265 |     return;
266 | }
267 |
268 | /*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
269 | /* end of getcmt.c  */
```

TEXT STATISTICS

```
  9772 characters
   269 lines
```

LEXICAL STATISTICS

```
    43 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
    18 constants [character]
    11 constants [string]
    24 constants [numeric]
```

SYMBOL TABLE

```
    EOF             178     183     243
    FILE             85
    LESS_FILLING             80     108     129     147     224     236
    LOOKS_GREAT              79     106     127     144     234
    argc             91     114     116
    argv             91     118     120     126     141     146     148
    ch              219     228     229     245     247
    chi             173     177     178     180     193     195     223     242     243     245     255
           259      260     263
    cht             173     182     183     185+    188     191     223     249     250     254
    count           174     176     184     187     197
    ctype            76
    ercode          124     126     134     136
    extract_c_cmts           82     162     171
    fopen           141
    fputs           107     128     145     146     235
    getc            177     182     195     242     249     263
    h        74      75      76      77
    help             96     128     131     132
    hype             94     107     110     111
    i        93     109     110     111     116+    118     120     126     130     131     132     141
           146     148     150     151     152     153     162     165     204     209+    211
    infile           85     141     142     177     182     191     195     242     249     254     263
    inside_c_cmt             83     188     219
    line_no          87     194     204     261
    lntoaz          202     235     237
    main             91
    num             204     207     211     212
    numbuf          205     208     211     214
    oops            103     145     151     152
    p       148     154     155     225     237     238     239
    putc            111     132     135     152     155     163     231     239     255     259
    show_nos         86     121     232
    stderr          107     111     128     132     135     145     146     152     155
    stdin            85
    stdio            75
    stdlib           74
    stdout          163     231     235     239     255     259
    string           77
    strncpy         208
    tolower         120
    ungetc          191     254
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** GETDCWD.C - returns the current working directory for a specific drive
**
** public domain by Bob Jarvis, modified by Bob Stout
** OS/2 compatibility mods by Ed Blackman
*/

#include "extkword.h"

#if defined(__OS2__)
 #define INCL_DOSFILEMGR    /* DosQueryCurrentDisk/Dir   */
 #define INCL_DOSMISC       /* DosError, DosQuerySysInfo */
 #include <os2.h>
 #define GetDrive(d) os2_getdrive(&d)

 void os2_getdrive(unsigned int *drive)
 {
       ULONG dummy;

       DosQueryCurrentDisk((ULONG *)&drive, &dummy);
 }

#elif defined(__ZTC__)
 #define GetDrive(d) dos_getdrive(&d)
#elif defined(__TURBOC__)
 #define GetDrive(d) ((d) = getdisk() + 1)
#else /* assume MSC */
 #define GetDrive(d) _dos_getdrive(&d)
#endif

#ifndef __OS2__
 #include <dos.h>
#endif      /* !__OS2__ */
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "dosfiles.h"

char *getdcwd(unsigned int drive)   /* 0 = current, 1 = A, 2 = B, etc */
{
      char *retptr, FAR *fretptr;
      unsigned long pathMaxLen = FILENAME_MAX + 4;

#ifdef __OS2__
      APIRET rc;
#else
      union REGS regs;
      struct SREGS sregs;
#endif


#ifdef __OS2__
/*
** it's bad practice in OS/2 to use hard coded values for things like
** the maximum length of paths and such.  Query the system instead.
*/
      rc = DosQuerySysInfo(QSV_MAX_PATH_LENGTH, QSV_MAX_PATH_LENGTH,
            &pathMaxLen, sizeof(pathMaxLen));
      if(0 != rc)
            return NULL;
#endif

      retptr = calloc((size_t)pathMaxLen, sizeof(char));
      if(retptr == NULL)
            return NULL;
      else  fretptr = (char FAR *)retptr;

      if(drive == 0)    /* figure out which drive is current */
            GetDrive(drive);

      *retptr = (char)((drive-1) + 'A');
      *(retptr+1) = ':';
      *(retptr+2) = '\\';

#ifdef __OS2__
      pathMaxLen -= 3;
      DosError(FERR_DISABLEHARDERR); /* disable "drive not ready" popups */

      rc = DosQueryCurrentDir(drive, retptr + 3, &pathMaxLen);

      DosError(FERR_ENABLEHARDERR);  /* re-enable popups */
#else
      segread(&sregs);
      regs.h.ah = 0x47;
      regs.h.dl = (unsigned char)drive;
      sregs.ds  = FP_SEG(fretptr);
      regs.x.si = FP_OFF(fretptr) + 3;

      intdosx(&regs, &regs, &sregs);
#endif

#ifdef __OS2__
      if (0 != rc)            /* drive number invalid or other error */
#else
      if (15 == regs.x.ax)    /* drive number invalid */
#endif
      {
            free(retptr);
```

```
 101 |             return NULL;
 102 |       }
 103 |       else  return retptr;
 104 | }
 105 |
 106 | #ifdef TEST
 107 |
 108 | main(int argc, char *argv[])
 109 | {
 110 |       char *curpath;
 111 |       unsigned int n;
 112 |
 113 |       if(argc > 1)
 114 |             n = (tolower(*argv[1]) - 'a') + 1;
 115 |       else  n = 0;
 116 |
 117 |       printf("curpath = '%s'\n", curpath = getdcwd(n));
 118 |       if (curpath)
 119 |             free(curpath);
 120 |       return 0;
 121 | }
 122 |
 123 | #endif /* TEST */
```

TEXT STATISTICS

```
  2975 characters
   123 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
    33 preprocessor instructions
     4 constants [character]
     3 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    APIRET          47
    DosError        79      83
    DosQueryCurrentDir              81
    DosQueryCurrentDisk             22
    DosQuerySysInfo         59
    FAR             43      68
    FERR_DISABLEHARDERR             79
    FERR_ENABLEHARDERR              83
    FILENAME_MAX            44
    FP_OFF          89
    FP_SEG          88
    GetDrive        16      26      28      30      71
    INCL_DOSFILEMGR         13
    INCL_DOSMISC            14
    NULL            62      66      67     101
    QSV_MAX_PATH_LENGTH             59+
    REGS            49
    SREGS           50
    TEST           106
    ULONG           20      22
    __OS2__         12      33      46      54      77      94
    __TURBOC__              27
    __ZTC__         25
    _dos_getdrive           30
    ah              86
    argc           108     113
    argv           108     114
    ax              97
    calloc          65
    ctype           38
    curpath        110     117     118     119
    d       16+     26+     28+     30+
    defined         12      25      27
    dl              87
    dos             34
    dos_getdrive            26
    drive           18      22      41      70      71      73      81      87
    ds              88
    dummy           20      22
    free           100     119
    fretptr         43      68      88      89
    getdcwd         41     117
    getdisk         28
    h       15      34      36      37      38      86      87
    intdosx         91
    main           108
    n      111     114     115     117
    os2             15
    os2_getdrive            16      18
    pathMaxLen              44      60+     65      78      81
    printf         117
    rc              47      59      61      81      95
    regs            49      86      87      89      91+     97
    retptr          43      65      66      68      73      74      75      81     100     103
    segread         85
    si              89
    size_t          65
    sregs           50      85      88      91
    stdio           37
```

```
stdlib           36
tolower         114
x       89       97
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************/
/*                                                                           */
/*  GETOPTS.C - Universal command line options parser                        */
/*                                                                           */
/*  Original Copyright 1990-96 by Robert B. Stout as part of                 */
/*  the MicroFirm Function Library (MFL)                                     */
/*                                                                           */
/*  The user is granted a free limited license to use this source file       */
/*  to create royalty-free programs, subject to the terms of the             */
/*  license restrictions specified in the LICENSE.MFL file.                  */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include "sniptype.h"
#include "filnames.h"
#include "errors.h"
#include "dirport.h"
#include "snipmath.h"
#include "minmax.h"
#include "getopts.h"
#include "numcnvrt.h"

#define MAX_XARGS 512

int         xargc = 0;
char       *xargv[MAX_XARGS]        = {NULL};
Boolean_T   getopts_range_err       = False_;
Boolean_T   xargs_on                = True_;

static FILE *rspfile = NULL;
static int   count = 0, argidx = 0;

enum proc_stat { PSerror = -1, PSok, PSliteral};

static Boolean_T      PASCAL bounds(struct Option_Tag *);
static enum proc_stat PASCAL swProc(char *swStr);
static void           PASCAL argProc(char *argStr);

/*
**  getopts()
**
**  Parameters: 1 - argc from main()
**              2 - argv from main()
**              3 - your program's options[] array
**
**  Returns: Number of options specified or Error_
**
**  Notes: 1. Your program should declare the global options[] array which
**            specifies all options recognized by getopts().
**
**         2. Out of range data are coerced into range and getopts_range_err
**            is set True_.
*/

int getopts(int argc, char *argv[])
{
      int i;
      Boolean_T scanning = True_;
      struct Option_Tag *ptr;
      char rspfname[FILENAME_MAX];
      char newarg[256];
      enum proc_stat ps;

      xargc = argc;
      xargv[argidx++] = argv[0];
      for (i = 1, count = 0; i < argc; )
      {
            /*
            ** If necessary, open a response file
            */

            if (scanning && !rspfile && '@' == argv[i][0])
            {
                  rspfile = cant(&argv[i][1], "r");
                  --xargc;
                  continue;
            }

            /*
            ** Get the next argument
            */

            if (rspfile)
            {
                  if (NULL == fgets(newarg, 256, rspfile))
                  {
                        rspfile = NULL;
                        ++i;
                        continue;
                  }
                  else
                  {
                        if ('\n' == LAST_CHAR(newarg))
```

```
101 |                                 LAST_CHAR(newarg) = NUL;
102 |                     }
103 |               }
104 |               else
105 |               {
106 |                     strcpy(newarg, argv[i++]);
107 |               }
108 |
109 |               /*
110 |               ** Per Unix tradition, back-to-back switch characters signify
111 |               ** the end of the switches
112 |               */
113 |
114 |               if ((2 == strlen(newarg)) && strchr("-/", newarg[0]) &&
115 |                     strchr("-/", newarg[1]))
116 |               {
117 |                     scanning = False_;
118 |                     if (!rspfile)
119 |                           --xargc;
120 |                     continue;
121 |               }
122 |
123 |               if (scanning && (strchr("-/", newarg[0])))
124 |               {
125 |                     ps = swProc(newarg);
126 |
127 |                     if (PSerror == ps)
128 |                           return Error_;
129 |
130 |                     if (PSok == ps)
131 |                           continue;
132 |               }
133 |
134 |               /*
135 |               ** If we got here, newarg must be an argument or filename
136 |               */
137 |
138 |               argProc(newarg);
139 |         }
140 |         return count;
141 | }
142 |
143 | /*
144 | **  Static function to process switch statements
145 | **
146 | **  Parameters: 1 - argv[i] containing the switch
147 | **
148 | **  Returns: PSok      if switch successful
149 | **           PSerror   if invalid
150 | **           PSliteral if literal (non-switch) argument
151 | */
152 |
153 | static enum proc_stat PASCAL swProc(char *swStr)
154 | {
155 |       struct Option_Tag *ptr;
156 |       Boolean_T searching;
157 |       unsigned short byte_var;
158 |       char *arg_ptr;
159 |
160 |       /*
161 |       ** Found a switch - If the 2nd character is also a switch
162 |       ** character. If so, then it's a literal and is skipped
163 |       */
164 |
165 |       if (strchr("-/@", swStr[1]))
166 |             return PSliteral;
167 |
168 |       for (ptr = options, searching = True_; searching; ++ptr)
169 |       {
170 |             if (!ptr->case_sense)
171 |             {
172 |                   ptr->letter = toupper(ptr->letter);
173 |                   swStr[1]    = toupper(swStr[1]);
174 |             }
175 |             if ((int)swStr[1] == ptr->letter) switch (ptr->type)
176 |             {
177 |             case Boolean_Tag:
178 |                   if ('-' == swStr[2])
179 |                         *((Boolean_T *)(ptr->buf)) = False_;
180 |                   else  *((Boolean_T *)(ptr->buf)) = True_;
181 |                   searching = False_;
182 |                   break;
183 |
184 |             case Byte_Tag:
185 |                   if (!swStr[2])
186 |                   {
187 |                         if (ptr->Default)
188 |                               arg_ptr = ptr->Default;
189 |                         else  return PSerror;
190 |                   }
191 |                   else  arg_ptr = &swStr[2];
192 |
193 |                   sscanf(arg_ptr, "%hx", &byte_var);
194 |                   *((char *)(ptr->buf)) = (unsigned char)(byte_var & 0xff);
195 |                   bounds(ptr);
196 |                   searching = False_;
197 |                   break;
198 |
199 |             case Int_Tag:
200 |                   if (!swStr[2])
```

```
201 |                   {
202 |                         if (ptr->Default)
203 |                               arg_ptr = ptr->Default;
204 |                         else  return PSerror;
205 |                   }
206 |                   else  arg_ptr = &swStr[2];
207 |
208 |                   *((int *)(ptr->buf)) = (int)(dround(getopts_eval(arg_ptr)));
209 |                   bounds(ptr);
210 |                   searching = False_;
211 |                   break;
212 |
213 |             case Short_Tag:
214 |                   if (!swStr[2])
215 |                   {
216 |                         if (ptr->Default)
217 |                               arg_ptr = ptr->Default;
218 |                         else  return PSerror;
219 |                   }
220 |                   else  arg_ptr = &swStr[2];
221 |
222 |                   *((short *)(ptr->buf)) =
223 |                         (short)(dround(getopts_eval(arg_ptr)));
224 |                   bounds(ptr);
225 |                   searching = False_;
226 |                   break;
227 |
228 |             case Word_Tag:
229 |                   if (!swStr[2])
230 |                   {
231 |                         if (ptr->Default)
232 |                               arg_ptr = ptr->Default;
233 |                         else  return PSerror;
234 |                   }
235 |                   else  arg_ptr = &swStr[2];
236 |
237 |                   sscanf(arg_ptr, "%hx", (unsigned short *)(ptr->buf));
238 |                   bounds(ptr);
239 |                   searching = False_;
240 |                   break;
241 |
242 |             case Long_Tag:
243 |                   if (!swStr[2])
244 |                   {
245 |                         if (ptr->Default)
246 |                               arg_ptr = ptr->Default;
247 |                         else  return PSerror;
248 |                   }
249 |                   else  arg_ptr = &swStr[2];
250 |
251 |                   *((long *)(ptr->buf)) =
252 |                         (long)(dround(getopts_eval(arg_ptr)));
253 |                   bounds(ptr);
254 |                   searching = False_;
255 |                   break;
256 |
257 |             case DWord_Tag:
258 |                   if (!swStr[2])
259 |                   {
260 |                         if (ptr->Default)
261 |                               arg_ptr = ptr->Default;
262 |                         else  return PSerror;
263 |                   }
264 |                   else  arg_ptr = &swStr[2];
265 |
266 |                   sscanf(arg_ptr, "%lx", (unsigned long *)(ptr->buf));
267 |                   bounds(ptr);
268 |                   searching = False_;
269 |                   break;
270 |
271 |             case Float_Tag:
272 |                   if (!swStr[2])
273 |                   {
274 |                         if (ptr->Default)
275 |                               arg_ptr = ptr->Default;
276 |                         else  return PSerror;
277 |                   }
278 |                   else  arg_ptr = &swStr[2];
279 |
280 |                   *((double *)(ptr->buf)) = (double)getopts_eval(arg_ptr);
281 |                   bounds(ptr);
282 |                   searching = False_;
283 |                   break;
284 |
285 |             case DFloat_Tag:
286 |                   if (!swStr[2])
287 |                   {
288 |                         if (ptr->Default)
289 |                               arg_ptr = ptr->Default;
290 |                         else  return PSerror;
291 |                   }
292 |                   else  arg_ptr = &swStr[2];
293 |
294 |                   *((double *)(ptr->buf)) = (double)getopts_eval(arg_ptr);
295 |                   bounds(ptr);
296 |                   searching = False_;
297 |                   break;
298 |
299 |             case String_Tag:
300 |                   if (!swStr[2] && ptr->Default)
```

```
301 |                       strcpy(ptr->buf, (char *)(ptr->Default));
302 |                 else  strcpy(ptr->buf, &swStr[2]);
303 |
304 |                 searching = False_;
305 |                 break;
306 |
307 |           default:
308 |                 return Error_;
309 |           }
310 |     }
311 |     ++count;
312 |     if (!rspfile)
313 |           --xargc;
314 |
315 |     return PSok;
316 | }
317 |
318 | /*
319 | **  Static function to process arguments
320 | */
321 |
322 | static void PASCAL argProc(char *argStr)
323 | {
324 |     DOSFileData ff;
325 |
326 |     /* If no wildcards or ignoring wildcards, just copy it */
327 |
328 |     if (!xargs_on || !has_wild(argStr))
329 |     {
330 |           xargv[argidx] = malloc(strlen(argStr) + 1);
331 |           if (NULL == xargv[argidx])
332 |                 ErrExit("Out of memory");
333 |           strcpy(xargv[argidx], argStr);
334 |           ++argidx;
335 |           return;
336 |     }
337 |     else  /* Expand wildcards, if possible                      */
338 |     {
339 |           if (Success_ == FIND_FIRST(argStr, _A_ANY, &ff))
340 |           {
341 |                 char path[FILENAME_MAX];
342 |                 char *p;
343 |
344 |                 /* Save the path for re-attachment             */
345 |
346 |                 fnSplit(argStr, NULL, path, NULL, NULL, NULL, NULL);
347 |
348 |                 --xargc;    /* We add stuff in the loop, so back up   */
349 |                 do
350 |                 {
351 |                       xargv[argidx] = malloc(strlen(ff_name(&ff))
352 |                                                 + strlen(path) + 2);
353 |                       if (NULL == xargv[argidx])
354 |                             ErrExit("Out of memory");
355 |                       fnMerge(xargv[argidx], NULL, path, NULL, ff_name(&ff),
356 |                               NULL, NULL);
357 |                       ++argidx;
358 |                       ++xargc;
359 |
360 |                 } while (Success_ == FIND_NEXT(&ff));
361 |                 FIND_END(&ff);
362 |           }
363 |     }
364 | }
365 |
366 | /*
367 | **  Assure new data are within specified ranges, return non-zero if coerced
368 | */
369 |
370 | static Boolean_T PASCAL bounds(struct Option_Tag *option)
371 | {
372 |     Boolean_T coerced = False_;
373 |     union {
374 |           unsigned char     B;
375 |           int               I;
376 |           short             S;
377 |           unsigned short    W;
378 |           long              L;
379 |           unsigned long     DW;
380 |           float             F;
381 |           double            D;
382 |     } tmp, val;
383 |
384 |     switch(option->type)
385 |     {
386 |     case Byte_Tag:
387 |           tmp.B = *((unsigned char *)(option->buf));
388 |           if (option->max)
389 |           {
390 |                 sscanf(option->max, "%hx", &val.B);
391 |                 tmp.B = min(tmp.B, val.B);
392 |           }
393 |           if (option->min)
394 |           {
395 |                 sscanf(option->min, "%hx", &val.B);
396 |                 tmp.B = max(tmp.B, val.B);
397 |           }
398 |           if (*((unsigned char *)(option->buf)) != tmp.B)
399 |           {
400 |                 getopts_range_err = True_;
```

```
401 |                   *((unsigned char *)(option->buf)) = tmp.B;
402 |                   coerced = True_;
403 |             }
404 |             break;
405 |
406 |       case Int_Tag:
407 |             tmp.I = *((int *)(option->buf));
408 |             if (option->max)
409 |             {
410 |                   val.D = dround(getopts_eval(option->max));
411 |                   if (val.D > (double)INT_MAX)
412 |                         val.I = INT_MAX;
413 |                   else  val.I = (int)val.D;
414 |                   tmp.I = min(tmp.I, val.I);
415 |             }
416 |             if (option->min)
417 |             {
418 |                   val.D = dround(getopts_eval(option->min));
419 |                   if (val.D < (double)INT_MIN)
420 |                         val.I = INT_MIN;
421 |                   else  val.I = (int)val.D;
422 |                   tmp.I = max(tmp.I, val.I);
423 |             }
424 |             if (*((int *)(option->buf)) != tmp.I)
425 |             {
426 |                   getopts_range_err = True_;
427 |                   *((int *)(option->buf)) = tmp.I;
428 |                   coerced = True_;
429 |             }
430 |             break;
431 |
432 |       case Short_Tag:
433 |             tmp.S = *((short *)(option->buf));
434 |             if (option->max)
435 |             {
436 |                   val.D = dround(getopts_eval(option->max));
437 |                   if (val.D > (double)SHRT_MAX)
438 |                         val.S = SHRT_MAX;
439 |                   else  val.S = (short)val.D;
440 |                   tmp.S = min(tmp.I, val.I);
441 |             }
442 |             if (option->min)
443 |             {
444 |                   val.D = dround(getopts_eval(option->min));
445 |                   if (val.D < (double)SHRT_MIN)
446 |                         val.S = SHRT_MIN;
447 |                   else  val.S = (short)val.D;
448 |                   tmp.S = max(tmp.I, val.I);
449 |             }
450 |             if (*((short *)(option->buf)) != tmp.S)
451 |             {
452 |                   getopts_range_err = True_;
453 |                   *((short *)(option->buf)) = tmp.I;
454 |                   coerced = True_;
455 |             }
456 |             break;
457 |
458 |       case Word_Tag:
459 |             tmp.W = *((unsigned short *)(option->buf));
460 |             if (option->max)
461 |             {
462 |                   sscanf(option->max, "%hx", &val.W);
463 |                   tmp.W = min(tmp.W, val.W);
464 |             }
465 |             if (option->min)
466 |             {
467 |                   sscanf(option->min, "%hx", &val.W);
468 |                   tmp.W = max(tmp.W, val.W);
469 |             }
470 |             if (*((unsigned short *)(option->buf)) != tmp.W)
471 |             {
472 |                   getopts_range_err = True_;
473 |                   *((unsigned short *)(option->buf)) = tmp.W;
474 |                   coerced = True_;
475 |             }
476 |             break;
477 |
478 |       case Long_Tag:
479 |             tmp.L = *((long *)(option->buf));
480 |             if (option->max)
481 |             {
482 |                   val.D = dround(getopts_eval(option->max));
483 |                   if (val.D > (double)LONG_MAX)
484 |                         val.L = LONG_MAX;
485 |                   else  val.L = (long)val.D;
486 |                   tmp.L = min(tmp.L, val.L);
487 |             }
488 |             if (option->min)
489 |             {
490 |                   val.D = dround(getopts_eval(option->min));
491 |                   if (val.D < (double)LONG_MIN)
492 |                         val.L = LONG_MIN;
493 |                   else  val.L = (int)val.D;
494 |                   tmp.L = max(tmp.L, val.L);
495 |             }
496 |             if (*((long *)(option->buf)) != tmp.L)
497 |             {
498 |                   getopts_range_err = True_;
499 |                   *((long *)(option->buf)) = tmp.L;
500 |                   coerced = True_;
```

```
501 |             }
502 |             break;
503 |
504 |       case DWord_Tag:
505 |             tmp.DW = *((unsigned long *)(option->buf));
506 |             if (option->max)
507 |             {
508 |                   sscanf(option->max, "%lx", &val.DW);
509 |                   tmp.DW = min(tmp.DW, val.DW);
510 |             }
511 |             if (option->min)
512 |             {
513 |                   sscanf(option->min, "%hx", &val.DW);
514 |                   tmp.DW = max(tmp.DW, val.DW);
515 |             }
516 |             if (*((unsigned long *)(option->buf)) != tmp.DW)
517 |             {
518 |                   getopts_range_err = True_;
519 |                   *((unsigned long *)(option->buf)) = tmp.DW;
520 |                   coerced = True_;
521 |             }
522 |             break;
523 |
524 |       case Float_Tag:
525 |             tmp.F = *((float *)(option->buf));
526 |             if (option->max)
527 |             {
528 |                   val.F = (float)getopts_eval(option->max);
529 |                   tmp.F = min(tmp.F, val.F);
530 |             }
531 |             if (option->min)
532 |             {
533 |                   val.F = (float)getopts_eval(option->min);
534 |                   tmp.F = max(tmp.F, val.F);
535 |             }
536 |             if (*((float *)(option->buf)) != tmp.F)
537 |             {
538 |                   getopts_range_err = True_;
539 |                   *((float *)(option->buf)) = tmp.F;
540 |                   coerced = True_;
541 |             }
542 |             break;
543 |
544 |       case DFloat_Tag:
545 |             tmp.D = *((double *)(option->buf));
546 |             if (option->max)
547 |             {
548 |                   val.D = getopts_eval(option->max);
549 |                   tmp.D = min(tmp.D, val.D);
550 |             }
551 |             if (option->min)
552 |             {
553 |                   val.D = getopts_eval(option->min);
554 |                   tmp.D = max(tmp.D, val.D);
555 |             }
556 |             if (*((double *)(option->buf)) != tmp.D)
557 |             {
558 |                   getopts_range_err = True_;
559 |                   *((double *)(option->buf)) = tmp.D;
560 |                   coerced = True_;
561 |             }
562 |             break;
563 |       }
564 |
565 |       return coerced;
566 | }
567 |
568 | /*
569 | **  Simplified evaluate() call - returns double or aborts
570 | */
571 |
572 | double getopts_eval(char *str)
573 | {
574 |       double retval;
575 |
576 |       if (Success_ == evaluate(str, &retval))
577 |             return retval;
578 |       else  ErrExit("Error evlauating \"%s\" - aborting\n", str);
579 | }
```

TEXT STATISTICS

18273 characters
579 lines

LEXICAL STATISTICS

27 comments [std-C]
0 comments [C++]
14 preprocessor instructions
3 constants [character]
25 constants [string]
42 constants [numeric]

SYMBOL TABLE

```
    B       374     387     390     391+    395     396+    398     401
    Boolean_T        34      35      42      65     156     179     180     370     372
    Boolean_Tag             177
    Byte_Tag        184     386
    D       381     410     411     413     418     419     421     436     437     439     444     445
            447     482     483     485     490     491     493     545     548     549+    553     554+
            556     559
    DFloat_Tag              285     544
    DOSFileData             324
    DW              379     505     508     509+    513     514+    516     519
    DWord_Tag       257     504
    Default         187     188     202     203     216     217     231     232     245     246     260
            261     274     275     288     289     300     301
    ErrExit         332     354     578
    Error_          128     308
    F       380     525     528     529+    533     534+    536     539
    FILE             37
    FILENAME_MAX             67     341
    FIND_END        361
    FIND_FIRST              339
    FIND_NEXT       360
    False_           34     117     179     181     196     210     225     239     254     268     282
            296     304     372
    Float_Tag       271     524
    I       375     407     412     413     414+    420     421     422+    424     427     440+    448+
            453
    INT_MAX         411     412
    INT_MIN         419     420
    Int_Tag         199     406
    L       378     479     484     485     486+    492     493     494+    496     499
    LAST_CHAR       100     101
    LONG_MAX        483     484
    LONG_MIN        491     492
    Long_Tag        242     478
    MAX_XARGS        30      33
    NUL             101
    NULL             33      37      92      94     331     346+    353     355+    356+
    Option_Tag               42      66     155     370
    PASCAL           42      43      44     153     322     370
    PSerror          40     127     189     204     218     233     247     262     276     290
    PSliteral        40     166
    PSok             40     130     315
    S       376     433     438     439     440     446     447     448     450
    SHRT_MAX        437     438
    SHRT_MIN        445     446
    Short_Tag       213     432
    String_Tag              299
    Success_        339     360     576
    True_            35      65     168     180     400     402     426     428     452     454     472
            474     498     500     518     520     538     540     558     560
    W       377     459     462     463+    467     468+    470     473
    Word_Tag        228     458
    _A_ANY          339
    argProc          44     138     322
    argStr           44     322     328     330     333     339     346
    arg_ptr         158     188     191     193     203     206     208     217     220     223     232
            235     237     246     249     252     261     264     266     275     278     280     289
            292     294
    argc             62      71      73
    argidx           38      72     330     331     333     334     351     353     355     357
    argv             62      72      79      81     106
    bounds           42     195     209     224     238     253     267     281     295     370
    buf             179     180     194     208     222     237     251     266     280     294     301
            302     387     398     401     407     424     427     433     450     453     459     470
            473     479     496     499     505     516     519     525     536     539     545     556
            559
    byte_var        157     193     194
    cant             81
    case_sense              170
    coerced         372     402     428     454     474     500     520     540     560     565
    count            38      73     140     311
    ctype            19
    dround          208     223     252     410     418     436     444     482     490
    evaluate        576
    ff              324     339     351     355     360     361
    ff_name         351     355
    fgets            92
    fnMerge         355
    fnSplit         346
    getopts          62
    getopts_eval            208     223     252     280     294     410     418     436     444     482
            490     528     533     548     553     572
    getopts_range_err        34     400     426     452     472     498     518     538     558
    h        16      17      18      19      20
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************/
/*                                                                           */
/*  MFLFILES Header file                                                     */
/*                                                                           */
/*  Function prototypes for file system access functions.                    */
/*                                                                           */
/*  Original Copyright 1990-96 by Robert B. Stout as part of                 */
/*  the MicroFirm Function Library (MFL)                                     */
/*                                                                           */
/*  The user is granted a free limited license to use this source file       */
/*  to create royalty-free programs, subject to the terms of the             */
/*  license restrictions specified in the LICENSE.MFL file.                  */
/*                                                                           */
/*****************************************************************************/

#ifndef GETOPTS_H
#define GETOPTS_H

#include <stdio.h>
#include "sniptype.h"
#include "extkword.h"

#if __cplusplus
 extern "C" {
#endif

/*****************************************************************************/
/*                                                                           */
/*  Process command line options and wildcard arguments                      */
/*                                                                           */
/*****************************************************************************/

typedef enum {
      Error_Tag,
      Boolean_Tag,
      Int_Tag,
      Short_Tag,
      Long_Tag,
      Byte_Tag,
      Word_Tag,
      DWord_Tag,
      Float_Tag,
      DFloat_Tag,
      String_Tag
      } TAG_TYPE;

struct Option_Tag {
      int         letter;                 /* Option switch              */
      Boolean_T   case_sense;             /* TRUE = case sensitive      */
      TAG_TYPE    type;                   /* Type of option             */
      void       *buf;                    /* Storage location           */
      char       *min;                    /* Minimum value (string)     */
      char       *max;                    /* Maximum value (string)     */
      char       *Default;                /* Default value (string)     */
};

extern struct Option_Tag options[];

extern int        xargc;
extern char      *xargv[];
extern Boolean_T  getopts_range_err;
extern Boolean_T  xargs_on;

int    getopts(int, char *[]);
double getopts_eval(char *str);

#if __cplusplus
 }
#endif

#endif /* GETOPTS_H */
```

```
TEXT STATISTICS

  2718 characters
    73 lines

LEXICAL STATISTICS

    28 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     3 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    Boolean_T       51      63      64
    Boolean_Tag             37
    Byte_Tag        41
    DFloat_Tag              45
    DWord_Tag       43
    Default         56
    Error_Tag       36
    Float_Tag       44
    GETOPTS_H       18      19
    Int_Tag         38
    Long_Tag        40
    Option_Tag              49      59
    Short_Tag       39
    String_Tag              46
    TAG_TYPE        47      52
    Word_Tag        42
    __cplusplus             25      69
    buf             53
    case_sense              51
    getopts         66
    getopts_eval            67
    getopts_range_err       63
    h       21
    letter          50
    max             55
    min             54
    options         59
    stdio           21
    str             67
    type            52
    xargc           61
    xargs_on        64
    xargv           62
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  GETOPTS.C - Universal command line options parser
**
**  Original Copyright 1993 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  This subset version is hereby donated to the public domain.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dirport.h"
#include "getoptsl.h"

#define NUL '\0'
#define MAX_XARGS 512

int   xargc;
char *xargv[MAX_XARGS];

/*
**  getopts()
**
**  Parameters: 1 - argc from main()
**              2 - argv from main()
**              3 - your program's options[] array
**
**  Returns: Number of options specified or -1 if error
**
**  Note: Your program should declare the global options[] array which
**        specifies all options recognized by getopts().
*/

int getopts_lite(int argc, char *argv[])
{
      int i, count, argidx = 0;
      char *argp;
      struct Option_Tag *ptr;

      xargc = argc;
      xargv[argidx++] = argv[0];
      for (i = 1, count = 0; i < argc; ++i)
      {
            if (strchr("-/", argv[i][0]) && !strchr("-/", argv[i][1]))
            {
                  Boolean_T found_it = False_;

                  /*
                  ** Found a switch - If the 2nd character is also a switch
                  ** character. If so, then it's a literal and is skipped
                  */

                  if (strchr("-/", argv[i][1]))
                        continue;

                  for (ptr = options; ptr->buf; ++ptr)
                  {
                        if ((int)argv[i][1] == ptr->letter) switch (ptr->type)
                        {
                        case Boolean_Tag:
                              if ('-' == argv[i][2])
                                    *((Boolean_T *)(ptr->buf)) = False_;
                              else  *((Boolean_T *)(ptr->buf)) = True_;
                              ++count;
                              --xargc;
                              found_it = True_;
                              break;

                        case Word_Tag:
                              sscanf(&argv[i][2], "%hd", (short *)(ptr->buf));
                              ++count;
                              --xargc;
                              found_it = True_;
                              break;

                        case DWord_Tag:
                              sscanf(&argv[i][2], "%ld", (long *)(ptr->buf));
                              ++count;
                              --xargc;
                              found_it = True_;
                              break;

                        case Double_Tag:
                              sscanf(&argv[i][2], "%lg",(double *)(ptr->buf));
                              ++count;
                              --xargc;
                              found_it = True_;
                              break;

                        case String_Tag:
                              strncpy(ptr->buf, &argv[i][2], ptr->siz - 1);
                              ++count;
                              --xargc;
                              found_it = True_;
                              break;

                        default:
```

```
                                return Error_;
                        }
                  }
                  if (!found_it)
                        return Error_;
            }
            else                          /* It must be a file name     */
            {
                  DOSFileData ffblk;

                  /* Set argp to point to the filename                  */

                  if (strchr("-/", argv[i][0]))
                        argp = &argv[i][1];
                  else  argp =  argv[i];

                  /* If no wildcards, just copy it */

                  if (!strchr(argp, '*') && !strchr(argp, '?'))
                  {
                        xargv[argidx++] = argp;
                        continue;
                  }

                  /* Expand wildcards, if possible                      */

                  if (0 == FIND_FIRST(argp, _A_ANY, &ffblk))
                  {
                        char path[FILENAME_MAX], *p;

                        /* Save the path for re-attachment              */

                        if (NULL == (p = strrchr(argp, '\\')))
                              p = strrchr(argp, '/');
                        if (p)
                        {
                              char ch = *p;

                              *p = NUL;
                              strcat(strcpy(path, argp), "\\");
                              *p = ch;
                        }
                        else  *path = NUL;
                        --xargc;
                        do
                        {
                              xargv[argidx] = malloc(strlen(ff_name(&ffblk))
                                    + strlen(path) + 2);
                              strcat(strcpy(xargv[argidx], path),
                                    ff_name(&ffblk));
                              ++argidx;
                              ++xargc;

                        } while (0 == FIND_NEXT(&ffblk));
                  }
            }
      }
      return count;
}

#ifdef TEST

#include <stdlib.h>
#include <conio.h>

Boolean_T test1 = True_, test2 = False_;
short test3     = -37;
long  test4     = 100000L;
char  test5[81] = "Default string";

struct Option_Tag options[] = {
      {'A',  Boolean_Tag, &test1, 0  },         /* Valid options        */
      {'B',  Boolean_Tag, &test2, 0  },
      {'C',  Word_Tag,    &test3, 0  },
      {'D',  DWord_Tag,   &test4, 0  },
      {'E',  String_Tag,   test5, 81 },
      {'\0', Error_Tag,     NULL, 0  }          /* Terminating record   */
};

#define TFprint(v) ((v) ? "TRUE" : "FALSE")

int main(int argc, char *argv[])
{
      int i;

      printf("Defaults:\ntest1 = %s\ntest2 = %s\ntest3 = %d\ntest4 = %ld\n"
            "test5 = \"%s\"\n\n", TFprint(test1), TFprint(test2), test3,
            test4, test5);

      printf("getopts() returned %d\n", getopts_lite(argc, argv));

      printf("Options are now:\ntest1 = %s\ntest2 = %s\ntest3 = %d\n"
            "test4 = %ld\ntest5 = \"%s\"\n\n", TFprint(test1),
            TFprint(test2), test3, test4, test5);

      puts("Hit any key to continue");
      getch();
      for (i = 0; i < xargc; ++i)
            printf("xargv[%d] = \"%s\"\n", i, xargv[i]);
      printf("\nxargc = %d\n", xargc);
```

```
 201 |       return 0;
 202 | }
 203 |
 204 | #endif /* TEST */


TEXT STATISTICS

  6715 characters
   204 lines

LEXICAL STATISTICS

    12 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
    12 constants [character]
    21 constants [string]
    31 constants [numeric]

SYMBOL TABLE

    Boolean_T       49      65      66     166
    Boolean_Tag             63     172     173
    DOSFileData            109
    DWord_Tag       79     175
    Double_Tag              86
    Error_         101     105
    Error_Tag      177
    FILENAME_MAX           129
    FIND_FIRST             127
    FIND_NEXT      154
    False_          49      65     166
    MAX_XARGS       19      22
    NUL             18     139     143
    NULL           133     177
    Option_Tag              41     171
    String_Tag              93     176
    TEST           161
    TFprint        180    187+     193     194
    True_           66      69      76      83      90      97     166
    Word_Tag        72     174
    _A_ANY         127
    argc            37      43      45     182     190
    argidx          39      44     121     147     149     151
    argp            40     114     115    119+     121     127     133     134     140
    argv            37      44     47+      56      61      64      73      80      87      94     113
           114     115     182     190
    buf             59      65      66      73      80      87      94
    ch             137     141
    conio          164
    count           39      45      67      74      81      88      95     158
    ff_name        147     150
    ffblk          109     127     147     150     154
    found_it        49      69      76      83      90      97     104
    getch          197
    getopts_lite            37     190
    h       12      13      14     163     164
    i       39     45+     47+      56      61      64      73      80      87      94     113     114
           115     184    198+    199+
    letter          61
    main           182
    malloc         147
    options         59     171
    p      129     133     134     135     137     139     141
    path           129     140     143     148     149
    printf         186     190     192     199     200
    ptr             41     59+     61+      65      66      73      80      87     94+
    puts           196
    siz             94
    sscanf          73      80      87
    stdio           12
    stdlib          13     163
    strcat         140     149
    strchr          47+     56     113    119+
    strcpy         140     149
    string          14
    strlen         147     148
    strncpy         94
    strrchr        133     134
    test1          166     172     187     193
    test2          166     173     187     194
    test3          167     174     187     194
    test4          168     175     188     194
    test5          169     176     188     194
    type            61
    v      180+
    xargc           21      43      68      75      82      89      96     144     152     198     200
    xargv           22      44     121     147     149     199
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  GETOPTS.H
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #ifndef GETOPTS__H
 10 | #define GETOPTS__H
 11 |
 12 | #include <stdlib.h>                     /* For size_t           */
 13 | #include "sniptype.h"
 14 |
 15 | typedef enum {
 16 |       Error_Tag = -1,
 17 |       Boolean_Tag,
 18 |       Word_Tag,
 19 |       DWord_Tag,
 20 |       Double_Tag,
 21 |       String_Tag
 22 |       } TAG_TYPE;
 23 |
 24 | struct Option_Tag {
 25 |       int         letter;               /* Option switch        */
 26 |       TAG_TYPE    type;                 /* Type of option       */
 27 |       void       *buf;                  /* Storage location     */
 28 |       size_t      siz;                  /* Size of buffer (used
 29 |                                            only for strings     */
 30 | };
 31 |
 32 | extern struct Option_Tag options[];
 33 | extern int    xargc;
 34 | extern char  *xargv[];
 35 |
 36 | int getopts_lite(int, char *[]);
 37 |
 38 | #endif /* GETOPTS__H */
```

TEXT STATISTICS

```
   914 characters
    38 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_Tag             17
    DWord_Tag       19
    Double_Tag              20
    Error_Tag       16
    GETOPTS__H               9      10
    Option_Tag              24      32
    String_Tag              21
    TAG_TYPE        22      26
    Word_Tag        18
    buf             27
    getopts_lite            36
    h       12
    letter          25
    options         32
    siz             28
    size_t          28
    stdlib          12
    type            26
    xargc           33
    xargv           34
```

```
/* +++Date last modified: 05-Jul-1997 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <float.h>
#include "sniptype.h"
#include "getopts.h"

Boolean_T     test1     = True_,
              test2     = False_;
int           test3     = -37;
long          test4     = 100000L;
double        test5;                    /* Initialized in main()      */
unsigned char test6     = 0x5a;
char          test7[80] = "Initial string";

char         test3min[]  = "-100";
char         test3max[]  = "32000";
char         test4min[]  = "-100000";
char         test4max[]  = "300000";
char         test5min[]  = "-12345.6789";
char         test5max[]  = "98765.4321";
char         test6max[]  = "ee";

char         test3dflt[] = "123";
char         test4dflt[] = "1234567";
char         test5dflt[] = "PI";
char         test6dflt[] = "a5";
char         test7dflt[] = "Default string";

struct Option_Tag options[] = {                 /* Valid options        */
      {
            'A',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            Boolean_Tag,                        /* Data type            */
            (void *)&test1,                     /* Storage address      */
            NULL,                               /* Range - min          */
            NULL,                               /* Range - max          */
            NULL                                /* Default value        */
      },
      {
            'B',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            Boolean_Tag,                        /* Data type            */
            (void *)&test2,                     /* Storage address      */
            NULL,                               /* Range - min          */
            NULL,                               /* Range - max          */
            NULL                                /* Default value        */
      },
      {
            'C',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            Int_Tag,                            /* Data type            */
            (void *)&test3,                     /* Storage address      */
            test3min,                           /* Range - min          */
            test3max,                           /* Range - max          */
            test3dflt                           /* Default value        */
      },
      {
            'D',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            Long_Tag,                           /* Data type            */
            (void *)&test4,                     /* Storage address      */
            test4min,                           /* Range - min          */
            test4max,                           /* Range - max          */
            test4dflt                           /* Default value        */
      },
      {
            'E',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            DFloat_Tag,                         /* Data type            */
            (void *)&test5,                     /* Storage address      */
            test5min,                           /* Range - min          */
            test5max,                           /* Range - max          */
            test5dflt                           /* Default value        */
      },
      {
            'F',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            Byte_Tag,                           /* Data type            */
            (void *)&test6,                     /* Storage address      */
            NULL,                               /* Range - min          */
            test6max,                           /* Range - max          */
            test6dflt                           /* Default value        */
      },
      {
            'G',                                /* Option letter        */
            False_,                             /* Case sensitiivity    */
            String_Tag,                         /* Data type            */
            (void *)test7,                      /* Storage address      */
            NULL,                               /* Range - min          */
            NULL,                               /* Range - max          */
            (void *)test7dflt                   /* Default value        */
      },
      {
            NUL,                                /* Terminating record   */
            False_,
```

```
101 |             Error_Tag,
102 |             NULL,
103 |             NULL,
104 |             NULL,
105 |             NULL
106 |       }
107 | };
108 |
109 | #define TFprint(v) ((v) ? "TRUE" : "FALSE")
110 |
111 | int main(int argc, char *argv[])
112 | {
113 |       int i;
114 |
115 |       test5 = getopts_eval("exp(1)");
116 |
117 |       printf("Defaults:\ntest1 = %s\ntest2 = %s\ntest3 = %d\ntest4 = %ld\n"
118 |             "test5 = %g\ntest6 = 0%2Xh\ntest7 = \"%s\"\n\n", TFprint(test1),
119 |             TFprint(test2), test3, test4, test5, test6, test7);
120 |
121 |       printf("getopts() returned %d\n", getopts(argc, argv));
122 |
123 |       printf("Options are now:\ntest1 = %s\ntest2 = %s\ntest3 = %d\n"
124 |             "test4 = %ld\ntest5 = %g\ntest6 = 0%2Xh\ntest7 = \"%s\"\n\n",
125 |             TFprint(test1), TFprint(test2), test3, test4, test5, test6,
126 |             test7);
127 |
128 |       puts("Hit any key to continue");
129 |       getch();
130 |       for (i = 0; i < xargc; ++i)
131 |             printf("xargv[%d] = \"%s\"\n", i, xargv[i]);
132 |       printf("\nxargc = %d\n", xargc);
133 |       return 0;
134 | }
```

TEXT STATISTICS

```
  5998 characters
   134 lines
```

LEXICAL STATISTICS

```
    53 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     7 constants [character]
    26 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       12
    Boolean_Tag             38      47
    Byte_Tag        83
    DFloat_Tag              74
    Error_Tag      101
    False_          13      37      46      55      64      73      82      91     100
    Int_Tag         56
    Long_Tag        65
    NUL             99
    NULL            40      41      42      49      50      51      85      94      95     102     103
           104     105
    Option_Tag              34
    String_Tag              92
    TFprint        109     118     119     125+
    True_           12
    argc           111     121
    argv           111     121
    conio            4
    getch          129
    getopts        121
    getopts_eval            115
    h        3       4       5       6       7       8
    i      113     130+    131+
    limits           7
    main           111
    math             6
    options         34
    printf         117     121     123     131     132
    puts           128
    stdio            3
    stdlib           5
    test1           12      39     118     125
    test2           13      48     119     125
    test3           14      57     119     125
    test3dflt       28      60
    test3max        21      59
    test3min        20      58
    test4           15      66     119     125
    test4dflt       29      69
    test4max        23      68
    test4min        22      67
    test5           16      75     115     119     125
    test5dflt       30      78
    test5max        25      77
    test5min        24      76
    test6           17      84     119     125
    test6dflt       31      87
    test6max        26      86
    test7           18      93     119     126
    test7dflt       32      96
    v      109+
    xargc          130     132
    xargv          131
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  GETSEG.H - How to get the memory segment of an object
**
**  public domain demo by Bob Stout
*/

#include <dos.h>
#include "extkword.h"

#define GetSeg(obj) (unsigned)((unsigned long)(((void FAR *)(obj))) >> 16)
#define GetOfs(obj) (unsigned)((unsigned long)(((void FAR *)(obj))) & 0xffff)

#ifdef TEST

#include <stdio.h>

int dummyv = 0;

int dummyf(void)
{
      return dummyv;
}

main()
{
      struct SREGS sregs;

      segread (&sregs);
      printf("DS = %04X, CS = %04X\n", sregs.ds, sregs.cs);

#if defined(__ZTC__)
      printf("&dummyv = %lp, dummyf = %lp\n\n",
#else
      printf("&dummyv = %Fp, dummyf = %Fp\n\n",
#endif

#if defined(__ZTC__) || defined(__TURBOC__)
            (int FAR *)&dummyv, (int (FAR *)())dummyf);
#else       /* MSC doesn't allow casting near function pointers to far  */
            (int FAR *)&dummyv, dummyf);
#endif

      printf("GetSeg(dummyv) = %04X, GetSeg(dummyf) = %04X\n",
            GetSeg(&dummyv), GetSeg(dummyf));
      printf("GetOfs(dummyv) = %04X, GetOfs(dummyf) = %04X\n",
            GetOfs(&dummyv), GetOfs(dummyf));
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1216 characters
    52 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     6 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    FAR             12      13      40+     42
    GetOfs          13      48+
    GetSeg          12      46+
    SREGS           28
    TEST            15
    __TURBOC__              39
    __ZTC__         33      39
    cs              31
    defined         33      39+
    dos              9
    ds              31
    dummyf          21      40      42      46      48
    dummyv          19      23      40      42      46      48
    h       9       17
    main            26
    obj             12+     13+
    printf          31      34      36      45      47
    segread         30
    sregs           28      30      31+
    stdio           17
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** GETSTRNG.C -- Demonstration of dynamic memory allocation to
**               receive string of unknown length.
**
** Ron Sires 1/31/89, released to the public domain.
** Bob Stout 2/18/93, modified to use a static buffer
*/

#include <stdlib.h>
#include <stdio.h>
#include "editgets.h"

#define BLOCKSIZ    16

char *getstring(void)
{
      int    newchar;
      size_t i;
      static size_t bufsize = 0;
      static char  *buffer  = NULL;

      /* Get chars from keyboard and put them in buffer.  */

      for (i = 0; ((newchar = getchar()) != EOF) && (newchar != '\n')
            && (newchar != '\r'); ++i )
      {
            if (i + 1 > bufsize)
            {
                  /* If buffer is full, resize it. */

                  if (NULL == (buffer = realloc(buffer, bufsize + BLOCKSIZ)))
                  {
                        puts("\agetstrng() - Insufficient memory");

                        /* Add terminator to partial string */

                        if (buffer)
                              buffer[i] = '\0';
                        return buffer;
                  }
                  bufsize += BLOCKSIZ;
            }
            buffer[i] = (char) newchar;
      }
      buffer[i] = '\0';   /* Tack on a null-terminator. */
      return buffer;
}

#ifdef TEST

#include <string.h>

int main(void)
{
      char *string;

      puts("Enter strings of any length or <Enter> to quit\n");
      do
      {
            string = getstring();
            printf("You entered:\n\"%s\"\n\n", string);
      } while (strlen(string));
      free(string);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1746 characters
    69 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     4 constants [character]
     4 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    BLOCKSIZ        15      33      43
    EOF             26
    EXIT_SUCCESS            66
    NULL            22      33
    TEST            51
    buffer          22      33+     39      40      41      45      47      48
    bufsize         21      29      33      43
    free            65
    getchar         26
    getstring       17      62
    h       11      12      53
    i       20      26      27      29      40      45      47
    main            55
    newchar         19      26+     27      45
    printf          63
    puts            35      59
    realloc         33
    size_t          20      21
    stdio           12
    stdlib          11
    string          53      57      62      63      64      65
    strlen          64
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  GETVOL.C - Retrieve a disk volume label
**             (proof you don't need FCBs to do it!)
**
**  public domain demo by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <io.h>

#include "dirport.h"
#include "sniptype.h"

char *getvol(char drive)
{
      char search[] = "A:\\*.*";
      static DOSFileData ff;

      *search = drive;
      if (Success_ == FIND_FIRST(search, _A_VOLID, &ff))
      {
            if (8 < strlen(ff_name(&ff)))       /* Eliminate period     */
                  strcpy(&(ff_name(&ff))[8], &(ff_name(&ff))[9]);
            return ff_name(&ff);
      }
      else  return NULL;
}

#ifdef TEST

int main(int argc, char *argv[])
{
      char *label;

      if (2 > argc)
      {
            puts("\aUsage: GETVOL d[:]");
            puts("where: d = drive letter (e.g. A, B, C, etc.");
            return -1;
      }
      if (NULL == (label = getvol(*argv[1])))
            printf("Unable to read a label on drive %c:\n", *argv[1]);
      else  printf("The volume label of drive %c: is \"%s\"\n",
            *argv[1], label);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1245 characters
    53 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     7 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    DOSFileData             22
    FIND_FIRST              25
    NULL            31      46
    Success_        25
    TEST            34
    _A_VOLID        25
    argc            36      40
    argv            36      46      47      49
    dos             13
    drive           19      24
    ff              22      25      27      28+     29
    ff_name         27      28+     29
    getvol          19      46
    h       10      11      12      13      14
    io              14
    label           38      46      49
    main            36
    printf          47      48
    puts            42      43
    search          21      24      25
    stdio           10
    stdlib          11
    strcpy          28
    string          12
    strlen          27
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************/
/*                                                                           */
/*  GETYN.C                                                                  */
/*                                                                           */
/*  Original Copyright 1993-94 by Robert B. Stout as part of                 */
/*  the MicroFirm Function Library (MFL)                                     */
/*                                                                           */
/*  The user is granted a free limited license to use this source file       */
/*  to create royalty-free programs, subject to the terms of the             */
/*  license restrictions specified in the LICENSE.MFL file.                  */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <time.h>
#include <ctype.h>
#include "sniptype.h"
#include "snipkbio.h"

/*****************************************************************************/
/*                                                                           */
/*  getYN()                                                                  */
/*                                                                           */
/*  Prompt a user for a Yes/No response with default and timeout             */
/*  features.                                                                */
/*                                                                           */
/*  Parameters: 1 - Prompt string                                            */
/*              2 - Default response, 'Y' or 'N', '\0' for none              */
/*              3 - Timeout before default assumed - 0 for none              */
/*                                                                           */
/*  Returns: True_ or False_                                                 */
/*                                                                           */
/*  NOTE: All output is via stderr                                           */
/*                                                                           */
/*****************************************************************************/

Boolean_T getYN(char *prompt, int def_ch, unsigned timeout)
{
      int ch;
      char *yn = " (y/n) ";
      clock_t start, limit = (clock_t)0;

      fputs(prompt, stderr);

      def_ch = toupper(def_ch);
      if ('Y' == def_ch)
            yn[2] = def_ch;
      else if ('N' == def_ch)
            yn[4] = def_ch;
      else  def_ch = '\0';

      fputs(yn, stderr);

      if (0 != def_ch && 0 < timeout)
      {
            start = clock();
            limit = (clock_t)(CLK_TCK * timeout);
      }
      while ('Y' != ch && 'N' != ch)
      {
            while (!kbhit())
            {
                  if (limit && (limit <= (clock() - start)))
                  {
                        ch = def_ch;
                        goto BYE;
                  }
            }
            ch = toupper(getch());
            if (def_ch && 'Y' != ch && 'N' != ch)
                  ch = def_ch;
      }

BYE:  fputc(ch, stderr);
      fputc('\n', stderr);
      return ('Y' == ch);
}
```

TEXT STATISTICS

```
  3315 characters
    79 lines
```

LEXICAL STATISTICS

```
    29 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     9 constants [character]
     3 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    BYE             68      76
    Boolean_T       39
    CLK_TCK         59
    ch              41      61+     67      71      72+     73      76      78
    clock           58      65
    clock_t         43+     59
    ctype           18
    def_ch          39      47+     48      49      50      51      52      56      67      72      73
    fputc           76      77
    fputs           45      54
    getYN           39
    getch           71
    h       16      17      18
    kbhit           63
    limit           43      59      65+
    prompt          39      45
    start           43      58      65
    stderr          45      54      76      77
    stdio           16
    time            17
    timeout         39      56      59
    toupper         47      71
    yn              42      49      51      54
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
   This code is based upon the program SETPATH.PAS (located in BPROGA) by
   David Dubois [71401,747]

   This Turbo C version is written by Peter Thomas [75716,2377]

   This series of routines are designed to Locate, Retrieve, Update, and
   Remove "Variables" from the MASTER copy of the DOS Environment table.
   The routines have been written in a manner that avoids linking any
   EXTERNAL routines for string manipulation, and thus should be independent
   of memory model being used.

   Be careful that changes made to the Environment with these routines
   ONLY OCCUR IN THE MASTER COPY, and that if COMMAND.COM is spawned
   from a routine that has changed the environment, NO CHANGES WILL BE
   SEEN IN THE ENVIRONMENT. This is most apparent when this program is run
   in the INTEGRATED environment: changes made by this technique will
   not appear if the "OS Shell" is invoked, and will only appear on exit
   from TC.

   For full documentation on the techniques used here can be found in the
   file COMENV.ARC located in LIB 2 of BPROGA on Compuserve.

   As David Dubois says:

  I hereby dedicate this knowledge to the public domain. Feel free to use
  it, but if you do, please mention my name. There are no guarantees, and
  in fact, I wouldn't bet a dollar that it will work every time.

  That this works at all is based on experimental, rather than properly
  documented, evidence. There are no guarantees. But then, its free.

*****************************************************************************/

#include <stdio.h>
#include <dos.h>
#include "extkword.h"
#include "mk_fp.h"

#if defined(__ZTC__) && (!defined(__SC__) || __SC__ < 0x700)
 #error ZTC, SC 6.x not supported - huge pointers required!
#endif

#ifdef __TURBOC__
 #include <alloc.h>
 #define Fmalloc farmalloc
 #define Ffree farfree
#else
 #include <malloc.h>
 #include <stdlib.h>
 #define Fmalloc _fmalloc
 #define Ffree _ffree
 #ifndef MK_FP
  #define MK_FP(seg,offset) \
        ((void far *)(((unsigned long)(seg)<<16) | (unsigned)(offset)))
 #endif
#endif

/*
 *   Mstr_FindEnvironment:
 *     Scans for the "Master" Environment area, and returns
 *     a pointer to it, and the size of the environment.
 */
void Mstr_FindEnvironment ( char FAR **Env , unsigned *EnvSize )
{
   unsigned int FAR *CommandSeg, FAR *TempSeg ;
   char FAR *BlockSeg ;

   /*
    *  Scan through PSP's looking for a block that is its own father.
    *  This block is the PSP of COMMAND.COM
    */
  TempSeg = MK_FP ( _psp , 0 ) ;
   do
   {
     CommandSeg = TempSeg ;
     TempSeg = MK_FP ( *(TempSeg+8) , 0 ) ;
   }
   while ( TempSeg != CommandSeg ) ;

   /*
    *  Scan forward through memory looking for the correct MSB.
    *  This will have COMMAND.COM's PSP as owner, and begin with
    *  the character M
    */
   BlockSeg = (char FAR *)CommandSeg ;
   do
   {
    BlockSeg = MK_FP ( FP_SEG(BlockSeg)+1 , 0 ) ;
   }
   while ( ( *(unsigned int FAR *)(BlockSeg+1) != FP_SEG ( CommandSeg ) ) ||
           ( *BlockSeg != 'M' ) ) ;

   /*
    *  The environment is the NEXT segment of memory
    *  and bytes 4 and 5 are the size in paragraphs
    */
   *Env = MK_FP ( FP_SEG(BlockSeg)+1 , 0 ) ;
```

```
101 |    *EnvSize = 16 * *(unsigned int FAR *)(BlockSeg+3) ;
102 | }
103 |
104 | /*
105 |  *   Mstr_getenv:
106 |  *     Scans the "Master" Environment for a given "sub string"
107 |  *     and returns a pointer to it.
108 |  *     Similar to Turbo routine "getenv" but uses the Master copy of the
109 |  *     environment table.
110 |  */
111 | char FAR *Mstr_getenv (char FAR *Env , char FAR *name)
112 | {
113 |    char FAR *Sub_Env, FAR *str1, FAR *str2 ;
114 |
115 |    /*
116 |     *  Start at the beginning of the environment
117 |     */
118 |    Sub_Env = Env ;
119 |
120 |    /*
121 |     *  While the "sub string" we're looking at is non-zero
122 |     */
123 |    for ( ; *Sub_Env ; )
124 |    {
125 |      /*
126 |       *  Simulate a "strcmp" on the "sub string" of the environment
127 |       *  and the string we're looking for
128 |       */
129 |      for ( str1 = Sub_Env , str2 = name ;
130 |            (*str1) && (*str2) && ( *str1 == *str2) ;
131 |            str1++ , str2++ ) ;
132 |      /*
133 |       *  If we reached the end of the string we're looking for
134 |       *  we've found the correct portion of the environment.
135 |       *  Return the ptr to the start of this "sub string"
136 |       */
137 |      if ( !*str2 )
138 |        return ( Sub_Env ) ;
139 |
140 |      /*
141 |       *  Otherwise, advance to the next "sub string" in the environment
142 |       *  by performing a "strchr" function
143 |       */
144 |      for ( ; *(Sub_Env++) ; ) ;
145 |    }
146 |
147 |    /*
148 |     *  Obviously, the string is not present in the environment.
149 |     *  Return this fact.
150 |     */
151 |  return ( NULL ) ;
152 | }
153 |
154 | /*
155 |  *   Mstr_delenv:
156 |  *     Scans the "Master" Environment for a given "sub string"
157 |  *     and removes it.
158 |  */
159 | int Mstr_delenv (char FAR *Env , unsigned EnvSize , char FAR *name)
160 | {
161 |    char FAR *Sub_Env , FAR *New_Env ;
162 |    char HUGE *Dest , FAR *Src , HUGE *End_Env ;
163 |
164 |    int Done  ;
165 |    unsigned Ctr ;
166 |
167 |    /*
168 |     *  Allocate a chunk of storage to act as a "working" copy of
169 |     *  the Environment table
170 |     */
171 |    New_Env = Fmalloc ( EnvSize ) ;
172 |
173 |    /*
174 |     *  Copy the data from the Master to Working copy of the
175 |     *  Environment table.
176 |     *  Simulates a "memcpy" function.
177 |     */
178 |    for ( Src = Env , Dest = (char FAR *)New_Env , Ctr = 0 ;
179 |          Ctr < EnvSize ;
180 |          *(Dest++) = *(Src++) , Ctr++ ) ;
181 |
182 |    /*
183 |     *  Scan the working copy of the environment for the desired
184 |     *  sub string
185 |     */
186 |    Sub_Env = Mstr_getenv ( New_Env , name ) ;
187 |
188 |    if ( Sub_Env == NULL )
189 |    {
190 |      /*
191 |       *  If not found, do nothing
192 |       */
193 |      Done = -1 ;
194 |    } else {
195 |      /*
196 |       *  Locate the end of the string to delete
197 |       *  Simulate a "strchr" call
198 |       */
199 |      for ( Src = Sub_Env ; *(Src++) ; ) ;
200 |
```

```
201 |      /*
202 |       *  Move the rest of the environment back over the "sub string"
203 |       *  being deleted.
204 |       *  Simulated "memcpy" function.
205 |       *  Huge pointers used for pointer comparison purposes.
206 |       */
207 |      for ( Dest = (char HUGE *)Sub_Env , End_Env = (char HUGE *) (New_Env + EnvSize ) ;
208 |          ( Dest < End_Env ) ;
209 |          *(Dest++) = *(Src++) ) ;
210 |
211 |      /*
212 |       *  Copy the data from the Working to Master copy of the
213 |       *  Environment table.
214 |       *  Simulates a "memcpy" function.
215 |       */
216 |      for ( Src = New_Env , Dest = (char HUGE *)Env , Ctr = 0 ;
217 |          Ctr < EnvSize ;
218 |          *(Dest++) = *(Src++) , Ctr++ ) ;
219 |
220 |      /*
221 |       *  Signal all done
222 |       */
223 |      Done = 0 ;
224 |    }
225 |
226 |    /*
227 |     *  Free all working storage
228 |     */
229 |    Ffree ( New_Env ) ;
230 |
231 |    return ( Done ) ;
232 | }
233 |
234 | /*
235 |  *   Mstr_putenv:
236 |  *     Adds/Replaces a given "sub string" in the Master Environment.
237 |  *     Similar to Turbo routine "putenv" but uses the Master copy of the
238 |  *     environment table.
239 |  */
240 | int Mstr_putenv (char FAR *Env , unsigned EnvSize , char FAR *name )
241 | {
242 |    char FAR *Sub_Env , FAR *Temp_Name ;
243 |    char HUGE *Dest , FAR *Src , HUGE *End_Env ;
244 |    int Done ;
245 |
246 |    /*
247 |     *  Allocate a chunk of storage to create the Variable name to add
248 |     *  to the Environment table
249 |     */
250 |    Temp_Name = Fmalloc ( 256 ) ;
251 |
252 |    /*
253 |     *  Extract only the Name portion of the data to add to the Environment
254 |     */
255 |    for ( Src = name , Dest = Temp_Name ;
256 |          *Src && ( *Src != '=' ) ;
257 |         *(Dest++) = *(Src++) ) ;
258 |
259 |    /*
260 |     *  Ensure that the resulting name is well formed.
261 |     */
262 |    *(Dest++) = '=' ;
263 |    *Dest = 0 ;
264 |
265 |    /*
266 |     *  Delete this sub string if found in the environment
267 |     */
268 |    Mstr_delenv ( Env , EnvSize , Temp_Name ) ;
269 |
270 |    /*
271 |     *  Locate the END of the Master table by locating a zero length
272 |     *  String in it
273 |     */
274 |    Sub_Env = Env ;
275 |    for ( ; *Sub_Env ; )
276 |    {
277 |      for ( ; *(Sub_Env++) ; ) ;
278 |    }
279 |
280 |    /*
281 |     *  Add the new string to the END of the existing environment, with
282 |     *  truncation IF needed
283 |     */
284 |    for ( Dest = (char HUGE *)(Sub_Env) , Src = name , End_Env = (char HUGE *) (Env + EnvSize ) ;
285 |          ( Dest < End_Env ) && (*Src) ;
286 |          *(Dest++) = *(Src++) ) ;
287 |
288 |    Done = -1 ;
289 |    if ( !*Src )
290 |    {
291 |      /*
292 |       *  If the string to add was FULLY added, ensure that the
293 |       *  newly updated environment is properly finished
294 |       */
295 |      Done = 0 ;
296 |      *(Dest++) = 0 ;
297 |      *Dest = 0 ;
298 |    }
299 |
300 |    /*
```

```
301 |     *  As a real safety measure, ensure that the FINAL two bytes of the
302 |     *  Environment are both 0. This will finish the last string AND
303 |     *  ensure that a zero length string is also present
304 |    */
305 |    *(End_Env-1) = 0 ;
306 |   *(End_Env-2) = 0 ;
307 |
308 |    /*
309 |     *  Free all working storage
310 |    */
311 |    Ffree ( Temp_Name ) ;
312 |
313 |    return ( Done ) ;
314 | }
315 |
316 | main()
317 | {
318 |   char FAR *Env ;
319 |    unsigned EnvSize ;
320 |
321 |    /*
322 |     *  Locate the Master Table
323 |    */
324 |    Mstr_FindEnvironment ( &Env, &EnvSize ) ;
325 |
326 |    /*
327 |     *  Describe what we've just found
328 |    */
329 |    printf ( "Env = %Fp Size = %u\n\n" , Env , EnvSize ) ;
330 |
331 |    /*
332 |     *  Search for, and display LOCATIONS of PATH, FRED and BERT
333 |    */
334 |    printf ( "Search for PATH= returns %Fp\n",  Mstr_getenv ( Env , "PATH=" ) ) ;
335 |    printf ( "Search for FRED= returns %Fp\n",  Mstr_getenv ( Env , "FRED=" ) ) ;
336 |    printf ( "Search for BERT= returns %Fp\n",  Mstr_getenv ( Env , "BERT=" ) ) ;
337 |
338 |    /*
339 |     *  Add FRED and BERT to the environment
340 |    */
341 |    Mstr_putenv ( Env , EnvSize , "FRED=fred" ) ;
342 |    Mstr_putenv ( Env , EnvSize , "BERT=bert" ) ;
343 |
344 |    printf ( "\nAdded FRED and BERT to environment\n" ) ;
345 |
346 |    /*
347 |     *  Search for, and display LOCATIONS of PATH ,FRED and BERT
348 |    */
349 |    printf ( "Search for PATH= returns %Fp\n",  Mstr_getenv ( Env , "PATH=" ) ) ;
350 |    printf ( "Search for FRED= returns %Fp\n",  Mstr_getenv ( Env , "FRED=" ) ) ;
351 |    printf ( "Search for BERT= returns %Fp\n",  Mstr_getenv ( Env , "BERT=" ) ) ;
352 |
353 |    /*
354 |     *  Delete FRED from the environment
355 |    */
356 |    Mstr_delenv ( Env , EnvSize , "FRED=" ) ;
357 |
358 |    printf ( "\nDeleted FRED= from the environment\n" ) ;
359 |
360 |    /*
361 |     *  Search for, and display LOCATIONS of PATH, FRED, and BERT
362 |    */
363 |    printf ( "Search for PATH= returns %Fp\n",  Mstr_getenv ( Env , "PATH=" ) ) ;
364 |    printf ( "Search for FRED= returns %Fp\n",  Mstr_getenv ( Env , "FRED=" ) ) ;
365 |    printf ( "Search for BERT= returns %Fp\n",  Mstr_getenv ( Env , "BERT=" ) ) ;
366 |
367 |    printf ( "\nIssue the DOS command SET and see that BERT is now set\n" ) ;
368 |
369 |    return 0;
370 | }
```

TEXT STATISTICS

```
 10564 characters
   370 lines
```

LEXICAL STATISTICS

```
    40 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
     3 constants [character]
    27 constants [string]
    28 constants [numeric]
```

SYMBOL TABLE

```
    BlockSeg        69      88      91+     93      94      100     101
    CommandSeg              68      78      81      88      93
    Ctr             165     178     179     180     216     217     218
    Dest            162     178     180     207     208     209     216     218     243     255     257
           262      263     284     285     286     296     297
    Done            164     193     223     231     244     288     295     313
    End_Env         162     207     208     243     284     285     305     306
    Env              66     100     111     118     159     178     216     240     268     274     284
           318      324     329     334     335     336     341     342     349     350     351     356
           363      364     365
    EnvSize          66     101     159     171     179     207     217     240     268     284     319
           324      329     341     342     356
    FAR              66      68+     69      88      93     101     111+    113+    159+    161+    162
           178      240+    242+    243     318
    FP_SEG           91      93     100
    Ffree            49      54     229     311
    Fmalloc          48      53     171     250
    HUGE            162+    207+    216     243+    284+
    MK_FP            55      56      75      79      91     100
    Mstr_FindEnvironment             66     324
    Mstr_delenv             159     268     356
    Mstr_getenv             111     186     334     335     336     349     350     351     363     364
           365
    Mstr_putenv             240     341     342
    NULL            151     188
    New_Env         161     171     178     186     207     216     229
    SC               43
    Src             162     178     180     199+    209     216     218     243     255     256+    257
           284      285     286     289
    Sub_Env         113     118     123     129     138     144     161     186     188     199     207
           242      274     275     277     284
    TempSeg          68      75      78      79+     81
    Temp_Name       242     250     255     268     311
    ZTC              43
    __SC__           42+
    __TURBOC__               46
    __ZTC__          42
    _ffree           54
    _fmalloc         53
    _psp             75
    alloc            47
    defined          42+
    dos              38
    far              57
    farfree          49
    farmalloc        48
    h        37      38      47      51      52
    huge             43
    main            316
    malloc           51
    name            111     129     159     186     240     255     284
    offset           56      57
    pointers         43
    printf          329     334     335     336     344     349     350     351     358     363     364
           365      367
    required         43
    seg              56      57
    stdio            37
    stdlib           52
    str1            113     129     130+    131
    str2            113     129     130+    131     137
    supported        43
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Demonstration of PC line drawing characters by David Harmon
  5 | */
  6 |
  7 | #include <stdio.h>
  8 |
  9 | main()
 10 | {
 11 |       /* compile and run this on a PC, and meditate on the results. */
 12 |
 13 |       puts("ASCII     Decimal       Hex            ASCII Dec Hex\n"
 14 |       "-----   -----------   --------         ----- --- ---\n"
 15 |       "\332 \302 \277   218 194 191   da c2 bf           \304   196  c4\n"
 16 |       "\303 \305 \264   195 197 180   c3 c5 b4           \263   179  b3\n"
 17 |       "\300 \301 \331   192 193 217   c0 c1 d9           \315   205  cd\n"
 18 |       "                                          \272   186  ba\n"
 19 |       "\311 \313 \273   201 203 187   c9 cb bb\n"
 20 |       "\314 \316 \271   204 206 185   cc ce b9           \260   176  b0\n"
 21 |       "\310 \312 \274   200 202 188   c8 ca bc           \261   177  b1\n"
 22 |       "                                          \262   178  b2\n"
 23 |       "\326 \322 \267   214 210 183   d6 d2 b7           \333   219  db\n"
 24 |       "\307 \327 \266   199 215 182   c7 d7 b6\n"
 25 |       "\323 \320 \275   211 208 189   d3 d0 bd           \334   220  dc\n"
 26 |       "                                          \335   221  dd\n"
 27 |       "\325 \321 \270   213 209 184   d5 d1 b8           \336   222  de\n"
 28 |       "\306 \330 \265   198 216 181   c6 d8 b5           \337   223  df\n"
 29 |       "\324 \317 \276   212 207 190   d4 cf be\n"
 30 |       );
 31 |
 32 |       return 0;
 33 | }
```

TEXT STATISTICS

```
  1429 characters
    33 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
    17 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    h        7
    main             9
    puts            13
    stdio            7
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * The  information  in  this  document  is  subject  to  change
 * without  notice  and  should not be construed as a commitment
 * by Digital Equipment Corporation or by DECUS.
 *
 * Neither Digital Equipment Corporation, DECUS, nor the authors
 * assume any responsibility for the use or reliability of  this
 * document or the described software.
 *
 *      Copyright (C) 1980, DECUS
 *
 * General permission to copy or modify, but not for profit,  is
 * hereby  granted,  provided that the above copyright notice is
 * included and reference made to  the  fact  that  reproduction
 * privileges were granted by DECUS.
 */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/*
 * grep
 *
 * Runs on the Decus compiler or on vms, On vms, define as:
 *      grep :== "$disk:[account]grep"      (native)
 *      grep :== "$disk:[account]grep grep" (Decus)
 * See below for more information.
 */

char    *documentation[] = {
"grep searches a file for a given pattern.  Execute by",
"   grep [flags] regular_expression file_list\n",
"Flags are single characters preceeded by '-':",
"   -c      Only a count of matching lines is printed",
"   -f      Print file name for matching lines switch, see below",
"   -n      Each line is preceeded by its line number",
"   -v      Only print non-matching lines\n",
"The file_list is a list of files (wildcards are acceptable on RSX modes).",
"\nThe file name is normally printed if there is a file given.",
"The -f flag reverses this action (print name no file, not if more).\n",
0 };

char    *patdoc[] = {
"The regular_expression defines the pattern to search for.  Upper- and",
"lower-case are always ignored.  Blank lines never match.  The expression",
"should be quoted to prevent file-name translation.",
"x      An ordinary character (not mentioned below) matches that character.",
"'\\'    The backslash quotes any character.  \"\\$\" matches a dollar-sign.",
"'^'    A circumflex at the beginning of an expression matches the",
"       beginning of a line.",
"'$'    A dollar-sign at the end of an expression matches the end of a line.",
"'.'    A period matches any character except \"new-line\".",
"':a'   A colon matches a class of characters described by the following",
"':d'     character.  \":a\" matches any alphabetic, \":d\" matches digits,",
"':n'     \":n\" matches alphanumerics, \": \" matches spaces, tabs, and",
"': '     other control characters, such as new-line.",
"'*'    An expression followed by an asterisk matches zero or more",
"       occurrances of that expression: \"fo*\" matches \"f\", \"fo\"",
"       \"foo\", etc.",
"'+'    An expression followed by a plus sign matches one or more",
"       occurrances of that expression: \"fo+\" matches \"fo\", etc.",
"'-'    An expression followed by a minus sign optionally matches",
"       the expression.",
"'[]'   A string enclosed in square brackets matches any character in",
"       that string, but no others.  If the first character in the",
"       string is a circumflex, the expression matches any character",
"       except \"new-line\" and the characters in the string.  For",
"       example, \"[xyz]\" matches \"xx\" and \"zyx\", while \"[^xyz]\"",
"       matches \"abc\" but not \"axb\".  A range of characters may be",
"       specified by two characters separated by \"-\".  Note that,",
"       [a-z] matches alphabetics, while [z-a] never matches.",
"The concatenation of regular expressions is a regular expression.",
0};

#define LMAX    512
#define PMAX    256

#define CHAR    1
#define BOL     2
#define EOL     3
#define ANY     4
#define CLASS   5
#define NCLASS  6
#define STAR    7
#define PLUS    8
#define MINUS   9
#define ALPHA   10
#define DIGIT   11
#define NALPHA  12
#define PUNCT   13
#define RANGE   14
#define ENDPAT  15

int cflag=0, fflag=0, nflag=0, vflag=0, nfile=0, debug=0;

char *pp, lbuf[LMAX], pbuf[PMAX];

extern char *cclass(), *pmatch();
```

```
101 |
102 |
103 | /*** Main program - parse arguments & grep *************/
104 | main(argc, argv)
105 | int argc;
106 | char *argv[];
107 | {
108 |    register char   *p;
109 |    register int    c, i;
110 |    int             gotpattern;
111 |
112 |    FILE            *f;
113 |
114 |    if (argc <= 1)
115 |       usage("No arguments");
116 |    if (argc == 2 && argv[1][0] == '?' && argv[1][1] == 0) {
117 |       help(documentation);
118 |       help(patdoc);
119 |       return 1;
120 |       }
121 |    nfile = argc-1;
122 |    gotpattern = 0;
123 |    for (i=1; i < argc; ++i) {
124 |       p = argv[i];
125 |       if (*p == '-') {
126 |          ++p;
127 |          while (c = *p++) {
128 |             switch(tolower(c)) {
129 |
130 |             case '?':
131 |                help(documentation);
132 |                break;
133 |
134 |             case 'C':
135 |             case 'c':
136 |                ++cflag;
137 |                break;
138 |
139 |             case 'D':
140 |             case 'd':
141 |                ++debug;
142 |                break;
143 |
144 |             case 'F':
145 |             case 'f':
146 |                ++fflag;
147 |                break;
148 |
149 |             case 'n':
150 |             case 'N':
151 |                ++nflag;
152 |                break;
153 |
154 |             case 'v':
155 |             case 'V':
156 |                ++vflag;
157 |                break;
158 |
159 |             default:
160 |                usage("Unknown flag");
161 |             }
162 |          }
163 |          argv[i] = 0;
164 |          --nfile;
165 |       } else if (!gotpattern) {
166 |          compile(p);
167 |          argv[i] = 0;
168 |          ++gotpattern;
169 |          --nfile;
170 |       }
171 |    }
172 |    if (!gotpattern)
173 |       usage("No pattern");
174 |    if (nfile == 0)
175 |       grep(stdin, 0);
176 |    else {
177 |       fflag = fflag ^ (nfile > 0);
178 |       for (i=1; i < argc; ++i) {
179 |          if (p = argv[i]) {
180 |             if ((f=fopen(p, "r")) == NULL)
181 |                cant(p);
182 |             else {
183 |                grep(f, p);
184 |                fclose(f);
185 |             }
186 |          }
187 |       }
188 |    }
189 |    return EXIT_SUCCESS;
190 | }
191 |
192 | /*** Display a file name ******************************/
193 | file(s)
194 | char *s;
195 | {
196 |    printf("File %s:\n", s);
197 | }
198 |
199 | /*** Report unopenable file ***************************/
200 | cant(s)
```

```
201 | char *s;
202 | {
203 |    fprintf(stderr, "%s: cannot open\n", s);
204 | }
205 |
206 | /*** Give good help **********************************/
207 | help(hp)
208 | char **hp;
209 | {
210 |    register char   **dp;
211 |
212 |    for (dp = hp; *dp; ++dp)
213 |       printf("%s\n", *dp);
214 | }
215 |
216 | /*** Display usage summary ****************************/
217 | usage(s)
218 | char    *s;
219 | {
220 |    fprintf(stderr, "?GREP-E-%s\n", s);
221 |    fprintf(stderr,
222 |       "Usage: grep [-cfnv] pattern [file ...].  grep ? for help\n");
223 |    exit(EXIT_FAILURE);
224 | }
225 |
226 | /*** Compile the pattern into global pbuf[] ************/
227 | compile(source)
228 | char       *source;   /* Pattern to compile */
229 | {
230 |    register char  *s;         /* Source string pointer     */
231 |    register char  *lp;        /* Last pattern pointer      */
232 |    register int   c;          /* Current character         */
233 |    int            o;          /* Temp                      */
234 |    char           *spp;       /* Save beginning of pattern */
235 |
236 |    s = source;
237 |    if (debug)
238 |       printf("Pattern = \"%s\"\n", s);
239 |    pp = pbuf;
240 |    while (c = *s++) {
241 |       /*
242 |        * STAR, PLUS and MINUS are special.
243 |        */
244 |       if (c == '*' || c == '+' || c == '-') {
245 |          if (pp == pbuf ||
246 |                (o=pp[-1]) == BOL ||
247 |                o == EOL ||
248 |                o == STAR ||
249 |                o == PLUS ||
250 |                o == MINUS)
251 |             badpat("Illegal occurrance op.", source, s);
252 |          store(ENDPAT);
253 |          store(ENDPAT);
254 |          spp = pp;               /* Save pattern end     */
255 |          while (--pp > lp)       /* Move pattern down    */
256 |             *pp = pp[-1];        /* one byte             */
257 |          *pp =   (c == '*') ? STAR :
258 |             (c == '-') ? MINUS : PLUS;
259 |          pp = spp;               /* Restore pattern end  */
260 |          continue;
261 |       }
262 |       /*
263 |        * All the rest.
264 |        */
265 |       lp = pp;         /* Remember start       */
266 |       switch(c) {
267 |
268 |       case '^':
269 |          store(BOL);
270 |          break;
271 |
272 |       case '$':
273 |          store(EOL);
274 |          break;
275 |
276 |       case '.':
277 |          store(ANY);
278 |          break;
279 |
280 |       case '[':
281 |          s = cclass(source, s);
282 |          break;
283 |
284 |       case ':':
285 |          if (*s) {
286 |             switch(tolower(c = *s++)) {
287 |
288 |             case 'a':
289 |             case 'A':
290 |                store(ALPHA);
291 |                break;
292 |
293 |             case 'd':
294 |             case 'D':
295 |                store(DIGIT);
296 |                break;
297 |
298 |             case 'n':
299 |             case 'N':
300 |                store(NALPHA);
```

```
                break;

             case ' ':
                store(PUNCT);
                break;

             default:
                badpat("Unknown : type", source, s);

             }
             break;
          }
          else    badpat("No : type", source, s);

       case '\\':
          if (*s)
             c = *s++;

       default:
          store(CHAR);
          store(tolower(c));
       }
    }
    store(ENDPAT);
    store(0);               /* Terminate string     */
    if (debug) {
       for (lp = pbuf; lp < pp;) {
          if ((c = (*lp++ & 0377)) < ' ')
             printf("\\%o ", c);
          else    printf("%c ", c);
        }
        printf("\n");
    }
}

/*** Compile a class (within []) **********************/
char *cclass(source, src)
char       *source;   /* Pattern start -- for error msg. */
char       *src;      /* Class start */
{
   register char   *s;        /* Source pointer    */
   register char   *cp;       /* Pattern start     */
   register int    c;         /* Current character */
   int             o;         /* Temp              */

   s = src;
   o = CLASS;
   if (*s == '^') {
      ++s;
      o = NCLASS;
   }
   store(o);
   cp = pp;
   store(0);                            /* Byte count      */
   while ((c = *s++) && c!=']') {
      if (c == '\\') {                  /* Store quoted char    */
         if ((c = *s++) == '\0')        /* Gotta get something  */
            badpat("Class terminates badly", source, s);
         else    store(tolower(c));
      }
      else if (c == '-' &&
            (pp - cp) > 1 && *s != ']' && *s != '\0') {
         c = pp[-1];             /* Range start     */
         pp[-1] = RANGE;         /* Range signal    */
         store(c);               /* Re-store start  */
         c = *s++;               /* Get end char and*/
         store(tolower(c));      /* Store it        */
      }
      else {
         store(tolower(c));      /* Store normal char */
      }
   }
   if (c != ']')
      badpat("Unterminated class", source, s);
   if ((c = (pp - cp)) >= 256)
      badpat("Class too large", source, s);
   if (c == 0)
      badpat("Empty class", source, s);
   *cp = c;
   return(s);
}

/*** Store an entry in the pattern buffer **************/
store(op)
   int op;
{
   if (pp >= &pbuf[PMAX])
      error("Pattern too complex\n");
   *pp++ = op;
}

/*** Report a bad pattern specification ***************/
badpat(message, source, stop)
char  *message;       /* Error message */
char  *source;        /* Pattern start */
char  *stop;          /* Pattern end   */
{
   fprintf(stderr, "-GREP-E-%s, pattern is\"%s\"\n", message, source);
   fprintf(stderr, "-GREP-E-Stopped at byte %d, '%c'\n",
         stop-source, stop[-1]);
```

```
401 |    error("?GREP-E-Bad pattern\n");
402 | }
403 |
404 | /*** Scan the file for the pattern in pbuf[] **********/
405 | grep(fp, fn)
406 | FILE       *fp;       /* File to process            */
407 | char       *fn;       /* File name (for -f option)  */
408 | {
409 |    register int lno, count, m;
410 |
411 |    lno = 0;
412 |    count = 0;
413 |    while (fgets(lbuf, LMAX, fp)) {
414 |       ++lno;
415 |       m = match();
416 |       if ((m && !vflag) || (!m && vflag)) {
417 |          ++count;
418 |          if (!cflag) {
419 |             if (fflag && fn) {
420 |                file(fn);
421 |                fn = 0;
422 |             }
423 |             if (nflag)
424 |                printf("%d\t", lno);
425 |             printf("%s\n", lbuf);
426 |          }
427 |       }
428 |    }
429 |    if (cflag) {
430 |       if (fflag && fn)
431 |          file(fn);
432 |       printf("%d\n", count);
433 |    }
434 | }
435 |
436 | /*** Match line (lbuf) with pattern (pbuf) return 1 if match ***/
437 | match()
438 | {
439 |    register char   *l;        /* Line pointer       */
440 |
441 |    for (l = lbuf; *l; ++l) {
442 |       if (pmatch(l, pbuf))
443 |          return(1);
444 |    }
445 |    return(0);
446 | }
447 |
448 | /*** Match partial line with pattern ******************/
449 | char *pmatch(line, pattern)
450 | char               *line;     /* (partial) line to match      */
451 | char               *pattern;  /* (partial) pattern to match   */
452 | {
453 |    register char   *l;        /* Current line pointer         */
454 |    register char   *p;        /* Current pattern pointer      */
455 |    register char   c;         /* Current character            */
456 |    char             *e;        /* End for STAR and PLUS match  */
457 |    int              op;        /* Pattern operation            */
458 |    int              n;         /* Class counter                */
459 |    char             *are;      /* Start of STAR match          */
460 |
461 |    l = line;
462 |    if (debug > 1)
463 |       printf("pmatch(\"%s\")\n", line);
464 |    p = pattern;
465 |    while ((op = *p++) != ENDPAT) {
466 |       if (debug > 1)
467 |          printf("byte[%d] = 0%o, '%c', op = 0%o\n",
468 |                l-line, *l, *l, op);
469 |       switch(op) {
470 |
471 |       case CHAR:
472 |          if (tolower(*l++) != *p++)
473 |             return(0);
474 |          break;
475 |
476 |       case BOL:
477 |          if (l != lbuf)
478 |             return(0);
479 |          break;
480 |
481 |       case EOL:
482 |          if (*l != '\0')
483 |             return(0);
484 |          break;
485 |
486 |       case ANY:
487 |          if (*l++ == '\0')
488 |             return(0);
489 |          break;
490 |
491 |       case DIGIT:
492 |          if ((c = *l++) < '0' || (c > '9'))
493 |             return(0);
494 |          break;
495 |
496 |       case ALPHA:
497 |          c = tolower(*l++);
498 |          if (c < 'a' || c > 'z')
499 |             return(0);
500 |          break;
```

```
501 |
502 |       case NALPHA:
503 |          c = tolower(*l++);
504 |          if (c >= 'a' && c <= 'z')
505 |             break;
506 |          else if (c < '0' || c > '9')
507 |             return(0);
508 |          break;
509 |
510 |       case PUNCT:
511 |          c = *l++;
512 |          if (c == 0 || c > ' ')
513 |             return(0);
514 |          break;
515 |
516 |       case CLASS:
517 |       case NCLASS:
518 |          c = tolower(*l++);
519 |          n = *p++ & 0377;
520 |          do {
521 |             if (*p == RANGE) {
522 |                p += 3;
523 |                n -= 2;
524 |                if (c >= p[-2] && c <= p[-1])
525 |                   break;
526 |             }
527 |             else if (c == *p++)
528 |                break;
529 |          } while (--n > 1);
530 |          if ((op == CLASS) == (n <= 1))
531 |             return(0);
532 |          if (op == CLASS)
533 |             p += n - 2;
534 |          break;
535 |
536 |       case MINUS:
537 |          e = pmatch(l, p);       /* Look for a match    */
538 |          while (*p++ != ENDPAT); /* Skip over pattern   */
539 |          if (e)                  /* Got a match?        */
540 |             l = e;               /* Yes, update string  */
541 |          break;                  /* Always succeeds     */
542 |
543 |       case PLUS:                 /* One or more ...     */
544 |          if ((l = pmatch(l, p)) == 0)
545 |             return(0);           /* Gotta have a match  */
546 |       case STAR:                 /* Zero or more ...    */
547 |          are = l;                /* Remember line start */
548 |          while (*l && (e = pmatch(l, p)))
549 |             l = e;               /* Get longest match   */
550 |          while (*p++ != ENDPAT); /* Skip over pattern   */
551 |          while (l >= are) {      /* Try to match rest   */
552 |             if (e = pmatch(l, p))
553 |                return(e);
554 |             --l;                 /* Nope, try earlier   */
555 |          }
556 |          return(0);              /* Nothing else worked */
557 |
558 |       default:
559 |          printf("Bad op code %d\n", op);
560 |          error("Cannot happen -- match\n");
561 |       }
562 |    }
563 |    return(l);
564 | }
565 |
566 | /*** Report an error *********************************/
567 | error(s)
568 | char *s;
569 | {
570 |    fprintf(stderr, "%s", s);
571 |    exit(EXIT_FAILURE);
572 | }
```

TEXT STATISTICS

16060 characters
572 lines

LEXICAL STATISTICS

74 comments [std-C]
0 comments [C++]
20 preprocessor instructions
51 constants [character]
71 constants [string]
81 constants [numeric]

SYMBOL TABLE

```
ALPHA           89     290     496
ANY             83     277     486
BOL             81     246     269     476
CHAR            80     320     471
CLASS           84     347     516     530     532
DIGIT           90     295     491
ENDPAT          94     252     253     324     465     538     550
EOL             82     247     273     481
EXIT_FAILURE           223     571
EXIT_SUCCESS           189
FILE           112     406
LMAX            77      98     413
MINUS           88     250     258     536
NALPHA          91     300     502
NCLASS          85     350     517
NULL           180
PLUS            87     249     258     543
PMAX            78      98     387
PUNCT           92     304     510
RANGE           93     364     521
STAR            86     248     257     546
are            459     547     551
argc           104     105     114     116     121     123     178
argv           104     106     116+    124     163     167     179
badpat         251     308     313     358     374     376     378     393
c      109     127     128     232     240     244+    257     258     266     286     317     321
       328     329     330     343     355+    356     357     359     361     363     365     366
       367     370     373     375     377     379     455     492+    497     498+    503     504+
       506+    511     512+    518     524+    527
cant           181     200
cclass         100     281     337
cflag           96     136     418     429
compile        166     227
count          409     412     417     432
cp             342     353     362     375     379
ctype           21
debug           96     141     237     326     462     466
documentation           32     117     131
dp             210     212+    213
e      456     537     539     540     548     549     552     553
error          388     401     560     567
exit           223     571
f      112     180     183     184
fclose         184
fflag           96     146     177+    419     430
fgets          413
file           193     420     431
fn             405     407     419     420     421     430     431
fopen          180
fp             405     406     413
fprintf        203     220     221     398     399     570
gotpattern             110     122     165     168     172
grep           175     183     405
h       19      20      21
help           117     118     131     207
hp             207     208     212
i      109     123+    124     163     167     178+    179
l      439     441+    442     453     461     468+    472     477     482     487     492     497
       503     511     518     537     540     544+    547     548+    549     551     552     554
       563
lbuf            98     413     425     441     477
line           449     450     461     463     468
lno            409     411     414     424
lp             231     255     265     327+    328
m      409     415     416+
main           104
match          415     437
message        393     394     398
n      458     519     523     529     530     533
nfile           96     121     164     169     174     177
nflag           96     151     423
o      233     246     247     248     249     250     344     347     350     352
op             384     385     389     457     465     468     469     530     532     559
p      108     124     125     126     127     166     179     180     181     183     454     464
       465     472     519     521     522     524+    527     533     537     538     544     548
       550     552
patdoc          45     118
pattern        449     451     464
pbuf            98     239     245     327     387     442
pmatch         100     442     449     537     544     548     552
pp              98     239     245     246     254     255     256+    257     259     265     327
       353     362     363     364     375     387     389
printf         196     213     238     329     330     332     424     425     432     463     467
       559
```

```
/* +++Date last modified: 05-Jul-1997 */

/* Written by Tom Torfs (2:292/516@fidonet,tomtorfs@dma.be)
   Donated to the public domain */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define /*_## (__\* /#*/__(/* #)#_*/_/*__ (#_)*/) /*__*/# _/*#_) _(_##_)\*#_*/
char _t[]=__(ABCDEF) __(GHIJKLM) __(NOPQRS) __(TUVWXYZ) __(abcdef) __(ghijklm)
__(nopqrs) __(tuvwxyz) ".,;- ?!''\n"; _(char *); _b(char); main(int c,char **a){
if (_(__(I##dt)),_(__(Q##Tqt)),_(__(d##gtc)),_(__(d##iti))) main(c-1,a+1);
if (_(__(dtQ##Tt)),_(__(s##ti)),_(__(W##Pi)),_(__(t##Xht))) main(c+1,a-1);
if (_(__(i##WT)),_(__(tfj##Th)),_(__(iXd##cp)),_(__(x##LW))) main(c-1,a+1);
if (_(__(TiW##Tg)),_(__(twi##Xh)),_(__(tc##dQ)),_(__(aT##gt))) main(c+1,a-1);
if (_(__(Xc##ti)),_(__(W##Tt)),_(__(b##Xc)),_(__(S##tid))) main(c-1,a+1);
if (_(__(t##hjU)),_(__(U##TgxI)),_(__(W##Tt)),_(__(h##aXcV))) main(c+1,a-1);
if (_(__(ht##Pc)),_(__(S##tPg)),_(__(g##dlht)),_(__(d##Utd))) main(c-1,a+1);
if (_(__(ji##gP)),_(__(V##Tdj)),_(__(h##tU)),_(__(d##gi))) main(c+1,a-1);
if (_(__(j##cTq)),_(__(x##Dgt)),_(__(id##ti)),_(__(P##ZT))) main(c-1,a+1);
if (_(__(t##Pg)),_(__(b##ht)),_(__(P##VPX)),_(__(c##hi))) main(c+1,a-1);
if (_(__(t##Pt)),_(__(hTP##td)),_(__(Uti##gd)),_(__(jQ##aT))) main(c-1,a+1);
if (_(__(h##qx)),_(__(y##cSt)),_(__(Q##ntd)),_(__(e##ed))) main(c+1,a-1);
if (_(__(hX##cV)),_(__(t##TcSt)),_(__(iW##Tb)),_(__(ut##st))) main(c-1,a+1);
if (_(__(I##dt)),_(__(S##XT)),_(__(t##sti)),_(__(d##tha))) main(c+1,a-1);
if (_(__(T##Tets)),_(__(xC##dtb)),_(__(dgT##rt)),_(__(Pc##St))) main(c-1,a+1);
if (_(__(Q##nt)),_(__(Pth##aT)),_(__(Te##ti)),_(__(dth##Pn))) main(c+1,a-1);
if (_(__(tl##TtT)),_(__(c##Sx)),_(__(I##WT)),_(__(t##WTP))) main(c-1,a+1);
if (_(__(g##isP)),_(__(RW##Tqt)),_(__(P##cSt)),_(__(i##WT))) main(c+1,a-1);
if (_(__(ti##Wdj)),_(__(hP##cS)),_(__(tc##Pi)),_(__(jgP##at))) main(c-1,a+1);
if (_(__(hW##dR)),_(__(Zh##xI)),_(__(W##Pi)),_(__(t##UaT))) main(c+1,a-1);
if (_(__(h##WtX)),_(__(ht##WT)),_(__(Xg##ti)),_(__(d##rtw))) main(c-1,a+1);
if (_(__(iXh##tP)),_(__(tR##dch)),_(__(jb##bP)),_(__(iX##dc))) main(c+1,a-1);
if (_(__(x)),_b(','),_(__(T##kd)),_(__(j##ia))) main(c-1,a+1);
if (_(__(n##tid)),_(__(tQT##tl)),_(__(XhW##TS)),_(__(p##tId))) main(c+1,a-1);
if (_(__(t##SXTt)),_(__(st##id)),_(__(tha##TT)),_(__(e##ts))) main(c-1,a+1);
if (_(__(xI##dth)),_(__(aT##Te)),_(__(v##te)),_(__(T##gR))) main(c+1,a-1);
if (_(__(WP##cR)),_(__(T##ti)),_(__(d##tSgT)),_(__(P##bp))) main(c-1,a+1);
if (_(__(ty##nq)),_(__(t##iWTg)),_(__(Tw##ht)),_(__(i##WTt))) main(c+1,a-1);
if (_(__(gj##Qq)),_(__(x)),_b('-'),_(__(dg##tX))) main(c-1,a+1);
if (_(__(cti##WP)),_(__(i##th)),_(__(a##TTet)),_(__(dU##tST))) main(c+1,a-1);
if (_(__(P##iW)),_(__(t##lW)),_(__(P##it)),_(__(S##gT))) main(c-1,a+1);
if (_(__(P##bh)),_(__(t##bPn)),_(__(t##Rdb)),_(__(T##qx))) main(c+1,a-1);
if (_(__(L##WT)),_(__(c##tlT)),_(__(t##WP)),_(__(k##TthW))) main(c-1,a+1);
if (_(__(j##UUa)),_(__(TS##tdU)),_(__(U##ti)),_(__(W##Xh))) main(c+1,a-1);
if (_(__(t##bd)),_(__(g##iPa)),_(__(t##RdXa)),_(__(q##xBjh))) main(c-1,a+1);
if (_(__(i##tV)),_(__(XkT##tj)),_(__(hte##Pj)),_(__(h##Tpt))) main(c+1,a-1);
if (_(__(IW##TgT)),_(__(wh##ti)),_(__(W##Ttg)),_(__(T##heT))) main(c-1,a+1);
if (_(__(R##ixIW)),_(__(P##it)),_(__(b##PZTh)),_(__(t##RPa))) main(c+1,a-1);
if (_(__(Pb##Xin)),_(__(t##dUt)),_(__(h##dta)),_(__(d##cV))) main(c-1,a+1);
if (_(__(t##aX)),_(__(U##Tp)),_(__(x)),_b('-')) main(c+1,a-1);
if (_(__(d##gt)),_(__(lW##dtl)),_(__(dja##St)),_(__(Q##TPg))) main(c-1,a+1);
if (_(__(t##iW)),_(__(T##tl)),_(__(WX##eh)),_(__(tPc##St))) main(c+1,a-1);
if (_(__(h##Rd)),_(__(gc##ht)),_(__(d##Uti)),_(__(X##bT))) main(c-1,a+1);
if (_(__(q##xIWT)),_(__(t##deeg)),_(__(Th##hd)),_(__(gw##htl))) main(c+1,a-1);
if (_(__(g##dc)),_(__(Vqt##iW)),_(__(T##tegd)),_(__(j##Stb))) main(c-1,a+1);
if (_(__(Pcw##ht)),_(__(Rd##ci)),_(__(jbT##an)),_(__(q##xIW))) main(c+1,a-1);
if (_(__(Tt##eP)),_(__(cV##ht)),_(__(d##UtS)),_(__(Th##eXh))) main(c-1,a+1);
if (_(__(TSt##ad)),_(__(k##Tq)),_(__(t##iWTt)),_(__(a##Pl))) main(c+1,a-1);
if (_(__(wh##tST)),_(__(a##Pn)),_(__(q##xI)),_(__(W##Tt))) main(c-1,a+1);
if (_(__(X##ch)),_(__(d##aTcR)),_(__(T##td)),_(__(U##tdU))) main(c+1,a-1);
if (_(__(UX##RTq)),_(__(t##PcS)),_(__(t##iWTt)),_(__(he##jg))) main(c-1,a+1);
if (_(__(ch##xI)),_(__(W##Pit)),_(__(e##Pi)),_(__(XT##ci))) main(c+1,a-1);
if (_(__(t##bT)),_(__(gXi##td)),_(__(Ut##iWT)),_(__(tjc##ld))) main(c-1,a+1);
if (_(__(g##iWnt)),_(__(i##PZT)),_(__(hq##xLW)),_(__(Tc##tWT))) main(c+1,a-1);
if (_(__(t##WXb)),_(__(h##TaU)),_(__(tb##XVW)),_(__(i##tW))) main(c-1,a+1);
if (_(__(X##htf)),_(__(j##XT)),_(__(ij##ht)),_(__(bPZ##Tx))) main(c+1,a-1);
if (_(__(L##XiW)),_(__(t##PtQP)),_(__(g##TtQd)),_(__(SZ##Xc))) main(c-1,a+1);
if (_(__(u##tL)),_(__(Wd##tl)),_(__(d##jaS)),_(__(t##UP))) main(c+1,a-1);
if (_(__(g##STah)),_(__(tQ##TPg)),_(__(qx##Idt)),_(__(V##gj))) main(c-1,a+1);
if (_(__(ci##tPc)),_(__(S##thlT)),_(__(P##it)),_(__(j##cS))) main(c+1,a-1);
if (_(__(Tgt##Pt)),_(__(lTP##gn)),_(__(t##aXU)),_(__(Tq##xzj))) main(c-1,a+1);
if (_(__(i##tiW)),_(__(Pit##iW)),_(__(T##tSg)),_(__(T##PS))) main(c+1,a-1);
if (_(__(t##dUt)),_(__(hdb##Ti)),_(__(W##Xc)),_(__(V##tP))) main(c-1,a+1);
if (_(__(Ui##Tg)),_(__(tS##TP)),_(__(i##Wt)),_(__(s##xIW))) main(c+1,a-1);
if (_(__(Tt##jcS)),_(__(Xh##Rdk)),_(__(TgT##St)),_(__(R##dj))) main(c-1,a+1);
if (_(__(c##ig)),_(__(nt##Ug)),_(__(d##btl)),_(__(W##dhT))) main(c+1,a-1);
if (_(__(t##Qdj)),_(__(gcx##Cd)),_(__(t##igP)),_(__(k##Ta))) main(c-1,a+1);
if (_(__(a##Tg)),_(__(t##gT)),_(__(i##jg)),_(__(c##hts))) main(c+1,a-1);
if (_(__(t##ejo)),_(__(o##aTh)),_(__(t##iW)),_(__(T##tlXa))) main(c-1,a+1);
if (_(__(aq##xyc)),_(__(S##tb)),_(__(PZ##Tht)),_(__(j##htg))) main(c+1,a-1);
if (_(__(PiW##Tg)),_(__(tQT##Pg)),_(__(t##iW)),_(__(d##hTt))) main(c-1,a+1);
if (_(__(X##aaht)),_(__(lTt##WP)),_(__(kTx##IW)),_(__(P##ct))) main(c+1,a-1);
if (_(__(U##an)),_(__(t##id)),_(__(tdi##WT)),_(__(g##ht))) main(c-1,a+1);
if (_(__(iW##Pit)),_(__(l##Tt)),_(__(Zc##dl)),_(__(t##cdi))) main(c+1,a-1);
if (_(__(td##Uu)),_(__(xI##Wj)),_(__(h##tRdc)),_(__(hR##XTc))) main(c-1,a+1);
if (_(__(R##Tt)),_(__(Sd##Th)),_(__(tb##PZ)),_(__(Tt##Rd))) main(c+1,a-1);
if (_(__(l##Pg)),_(__(Sh##td)),_(__(U##tj)),_(__(h##tP))) main(c-1,a+1);
if (_(__(a##aqx)),_(__(ycS##ti)),_(__(W##jhti)),_(__(W##Tt))) main(c+1,a-1);
if (_(__(c##Pi)),_(__(X##kTt)),_(__(W##jT)),_(__(td##Utg))) main(c-1,a+1);
if (_(__(T##hd)),_(__(a##ji)),_(__(X##dcx)),_b('!')) main(c+1,a-1);
if (_(__(h##th)),_(__(X##RZaX)),_(__(TS##tdw)),_(__(T##gt))) main(c-1,a+1);
if (_(__(l##XiWt)),_(__(i##WT)),_(__(teP##aT)),_(__(tR##Phi))) main(c+1,a-1);
if (_(__(t##dU)),_(__(ti##Wd)),_(__(jVW##ir)),_(__(x##yc))) main(c-1,a+1);
if (_(__(S##tT)),_(__(ciT##ge)),_(__(g##XhT)),_(__(h##tdUt))) main(c+1,a-1);
if (_(__(VgT##Pi)),_(__(teX##iR)),_(__(W##tP)),_(__(c##St))) main(c-1,a+1);
if (_(__(b##db)),_(__(T##ci)),_(__(qxL##Xi)),_(__(W##tiW))) main(c+1,a-1);
if (_(__(X##ht)),_(__(g##TV)),_(__(P##gS)),_(__(qt##iWT))) main(c-1,a+1);
if (_(__(Xg##tRj)),_(__(gg##Tc)),_(__(i##ht)),_(__(i##jgc))) main(c+1,a-1);
if (_(__(tP##lg)),_(__(nqx##yc)),_(__(S##ta)),_(__(d##hTti))) main(c-1,a+1);
```

```
 101 | if (_(__(W##Ttc)),_(__(P##bT)),_(__(td##Ut)),_(__(P##Ri))) main(c+1,a-1);
 102 | return _(__(Xd##cp)),_(__(x));} _(char *_a){int _c=atoi(__(_a));for (;*_a;_a++)
 103 | _c+=_b(*_a);return _c;}_b(char _a){char *_p=strchr(_t,_a);return
 104 | _p?!putchar(_t[((_p-_t)+11)%61]):atoi(__(_p));}


TEXT STATISTICS

  7532 characters
   104 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     4 constants [character]
     1 constants [string]
   182 constants [numeric]

SYMBOL TABLE

    ABCDEF          10
    Cd              78
    Dgt             20
    GHIJKLM         10
    I       12      25      28
    IW              48      83
    Idt             70
    L       44      68
    LW              14
    NOPQRS          10
    P       20      21      29      38      42+     43      49      71      83      98      101+
    PS              73
    PZ              81      87
    PZT             65
    PZTh            49
    Pb              50
    Pc              18      26
    PcS             62
    Pcw             57
    Pg              21      82      88
    Phi             93
    Pi              13      30      31      63      90      96
    PiW             82
    Pit             63      73      85
    Pj              47
    Pl              59
    Pn              27      60
    Pt              22      72
    PtQP            68
    Pth             27
    Q       12      23      27      52
    Qdj             78
    Qq              40
    R       49      76      87
    RPa             49
    RTq             62
    RW              29
    RZaX            92
    Rd              54      57      87
    RdXa            46
    Rdb             43
    Rdk             76
    Ri             101
    S       16      18      25      42      71      81      95     100
    STah            70
    SXTt            36
    SZ              68
    Sd              87
    Sh              88
    St              26      52      53      76      96
    Stb             56
    Sx              28
    T       26      34      37      38      43      48      53      56      61      73+     80      91
            92      97
    TP              75
    TPg             52      70
    TS              35      45      92
    TSt             59
    TT              36
    TTet            41
    TUVWXYZ         10
    TV              98
    Ta              78
    TaU             66
    Tb              24
    Tc              65      99
    TcSt            24
    Tdj             19
    Te              27      37
    Tets            26
    Tg              15      75      79      82
    TgT             48      76
    Tgt             72
    TgxI            17
    Th              14      55      58      87
    Tht             81
    Ti              74
    TiW             15
```

```
Tp              51
Tpt             47
Tq              59      72
Tqt             12      29
Tt              13      16      17      58      60      76      85      87+     89
TtQd            68
TtT             28
Ttc            101
Ttg             48
TthW            44
Tw              39
Tx              67
U       17      45      51      61      84      88
UP              69
UUa             45
UX              62
UaT             31
Ug              77
Ui              75
Ut              64     101
UtS             58
Utd             18
Utg             90
Uti             22      54
Uu              86
V       19      70      74
VPX             21
VgT             96
Vqt             56
W       13      16      17      31      45      48      60      63      74      77      89+     90
        96      97     101
WP              38      41      44      83
WT              14      28      29      32      44      84      93
WTP             28
WTt             39
WX              53
WXb             66
Wd              69      94
Wdj             30
Wj              86
Wt              75
WtX             32
X       54      61      67      83      90      91      92      98
XT              25      63      67
XTc             86
XVW             66
Xc              16+     68      74
Xd             102
Xg              32      99
Xh              15      45      76
XhT             95
XhW             35
Xht             13
Xi              97
XiW             68
XiWt            93
Xin             50
XkT             47
ZT              20
Zc              85
Zh              31
_        9+     11      12+     13+     14+     15+     16+     17+     18+     19+     20+     21+
        22+     23+     24+     25+     26+     27+     28+     29+     30+     31+     32+     33+
        34+     35+     36+     37+     38+     39+     40+     41+     42+     43+     44+     45+
        46+     47+     48+     49+     50+     51+     52+     53+     54+     55+     56+     57+
        58+     59+     60+     61+     62+     63+     64+     65+     66+     67+     68+     69+
        70+     71+     72+     73+     74+     75+     76+     77+     78+     79+     80+     81+
        82+     83+     84+     85+     86+     87+     88+     89+     90+     91+     92+     93+
        94+     95+     96+     97+     98+     99+    100+    101+    102+
__               9      10+     11+     12+     13+     14+     15+     16+     17+     18+     19+
        20+     21+     22+     23+     24+     25+     26+     27+     28+     29+     30+     31+
        32+     33+     34+     35+     36+     37+     38+     39+     40+     41+     42+     43+
        44+     45+     46+     47+     48+     49+     50+     51+     52+     53+     54+     55+
        56+     57+     58+     59+     60+     61+     62+     63+     64+     65+     66+     67+
        68+     69+     70+     71+     72+     73+     74+     75+     76+     77+     78+     79+
        80+     81+     82+     83+     84+     85+     86+     87+     88+     89+     90+     91+
        92+     93+     94+     95+     96+     97+     98+     99+    100+    101+    102+    104
_a             102+    103+
_b              11      34      40      51      91     103+
_c             102     103+
_p             103     104+
_t              10     103     104+
a       11      12      13      14      15      16      17      18      19      20      21      22
        23      24      25      26      27      28      29      30      31      32      33      34
        35      36      37      38      39      40     41+      42      43      44      45      46
        47      48      49      50      51      52      53      54      55      56      57      58
        59+     60+     61      62      63      64      65      66      67      68      69      70
        71      72      73      74      75      76      77      78     79+      80      81      82
        83      84      85      86      87      88     89+      90      91+     92      93      94
        95      96      97      98      99     100     101
aT              15      22      27      37      93
aTcR            61
aTh             80
aX              51
aXU             72
aXcV            17
aaht            83
abcdef          10
ad              59
an              57      84
```

```
ta              100
tb               66      81      87
tc               15      30
td               22      61      64      86      88      90     101
tdU              45      61
tdUt             95
tdi              84
tdw              92
te               37
teP              93
teX              96
tegd             56
tfj              14
th               41      92
tha              25      36
thlT             71
ti               13      16      20      27      30      32      38      45      48      89      94
tiW              73      97
tid              16      35
tj               47      88
tjc              64
tl               28      35      53      69
tlT              44
tlXa             80
ts               36
tuvwxyz          11
twi              15
ty               39
u       69
ut               24
v       37
wh               48      60
x       14       20      34      40      51      94     102
xBjh             46
xC               26
xI               31      37      60      63      86
xIW              57      75
xIWT             55
xLW              65
xyc              81
xzj              72
y       23
yc               94     100
ycS              89
```

```
/* +++Date last modified: 05-Jul-1997 */

#include <string.h>
#include <stdlib.h>

#include "hash.h"

/*
** public domain code by Jerry Coffin, with improvements by HenkJan Wolthuis.
**
** Tested with Visual C 1.0 and Borland C 3.1.
** Compiles without warnings, and seems like it should be pretty
** portable.
*/

/*
** These are used in freeing a table.  Perhaps I should code up
** something a little less grungy, but it works, so what the heck.
*/

static void (*function)(void *) = (void (*)(void *))NULL;
static hash_table *the_table = NULL;


/* Initialize the hash_table to the size asked for.  Allocates space
** for the correct number of pointers and sets them to NULL.  If it
** can't allocate sufficient memory, signals error by setting the size
** of the table to 0.
*/

hash_table *construct_table(hash_table *table, size_t size)
{
      size_t i;
      bucket **temp;

      table -> size  = size;
      table -> table = (bucket * *)malloc(sizeof(bucket *) * size);
      temp = table -> table;

      if ( temp == NULL )
      {
            table -> size = 0;
            return table;
      }

      for (i=0;i<size;i++)
            temp[i] = NULL;
      return table;
}


/*
** Hashes a string to produce an unsigned short, which should be
** sufficient for most purposes.
*/

static unsigned hash(char *string)
{
      unsigned ret_val = 0;
      int i;

      while (*string)
      {
            i = *( int *)string;
            ret_val ^= i;
            ret_val <<= 1;
            string ++;
      }
      return ret_val;
}

/*
** Insert 'key' into hash table.
** Returns pointer to old data associated with the key, if any, or
** NULL if the key wasn't in the table previously.
*/

void *insert(char *key, void *data, hash_table *table)
{
      unsigned val = hash(key) % table->size;
      bucket *ptr;

      /*
      ** NULL means this bucket hasn't been used yet.  We'll simply
      ** allocate space for our new bucket and put our data there, with
      ** the table pointing at it.
      */

      if (NULL == (table->table)[val])
      {
            (table->table)[val] = (bucket *)malloc(sizeof(bucket));
            if (NULL==(table->table)[val])
                  return NULL;

            (table->table)[val] -> key = strdup(key);
            (table->table)[val] -> next = NULL;
            (table->table)[val] -> data = data;
            return (table->table)[val] -> data;
      }

```

```
101 |       /*
102 |       ** This spot in the table is already in use.  See if the current string
103 |       ** has already been inserted, and if so, increment its count.
104 |       */
105 |
106 |       for (ptr = (table->table)[val];NULL != ptr; ptr = ptr -> next)
107 |             if (0 == strcmp(key, ptr->key))
108 |             {
109 |                   void *old_data;
110 |
111 |                   old_data = ptr->data;
112 |                   ptr -> data = data;
113 |                   return old_data;
114 |             }
115 |
116 |       /*
117 |       ** This key must not be in the table yet.  We'll add it to the head of
118 |       ** the list at this spot in the hash table.  Speed would be
119 |       ** slightly improved if the list was kept sorted instead.  In this case,
120 |       ** this code would be moved into the loop above, and the insertion would
121 |       ** take place as soon as it was determined that the present key in the
122 |       ** list was larger than this one.
123 |       */
124 |
125 |       ptr = (bucket *)malloc(sizeof(bucket));
126 |       if (NULL==ptr)
127 |             return 0;
128 |       ptr -> key = strdup(key);
129 |       ptr -> data = data;
130 |       ptr -> next = (table->table)[val];
131 |       (table->table)[val] = ptr;
132 |       return data;
133 | }
134 |
135 |
136 | /*
137 | ** Look up a key and return the associated data.  Returns NULL if
138 | ** the key is not in the table.
139 | */
140 |
141 | void *lookup(char *key, hash_table *table)
142 | {
143 |       unsigned val = hash(key) % table->size;
144 |       bucket *ptr;
145 |
146 |       if (NULL == (table->table)[val])
147 |             return NULL;
148 |
149 |       for ( ptr = (table->table)[val];NULL != ptr; ptr = ptr->next )
150 |       {
151 |             if (0 == strcmp(key, ptr -> key ) )
152 |                   return ptr->data;
153 |       }
154 |       return NULL;
155 | }
156 |
157 | /*
158 | ** Delete a key from the hash table and return associated
159 | ** data, or NULL if not present.
160 | */
161 |
162 | void *del(char *key, hash_table *table)
163 | {
164 |       unsigned val = hash(key) % table->size;
165 |       void *data;
166 |       bucket *ptr, *last = NULL;
167 |
168 |       if (NULL == (table->table)[val])
169 |             return NULL;
170 |
171 |       /*
172 |       ** Traverse the list, keeping track of the previous node in the list.
173 |       ** When we find the node to delete, we set the previous node's next
174 |       ** pointer to point to the node after ourself instead.  We then delete
175 |       ** the key from the present node, and return a pointer to the data it
176 |       ** contains.
177 |       */
178 |
179 |       for (last = NULL, ptr = (table->table)[val];
180 |             NULL != ptr;
181 |             last = ptr, ptr = ptr->next)
182 |       {
183 |             if (0 == strcmp(key, ptr -> key))
184 |             {
185 |                   if (last != NULL )
186 |                   {
187 |                         data = ptr -> data;
188 |                         last -> next = ptr -> next;
189 |                         free(ptr->key);
190 |                         free(ptr);
191 |                         return data;
192 |                   }
193 |
194 |                   /*
195 |                   ** If 'last' still equals NULL, it means that we need to
196 |                   ** delete the first node in the list. This simply consists
197 |                   ** of putting our own 'next' pointer in the array holding
198 |                   ** the head of the list.  We then dispose of the current
199 |                   ** node as above.
200 |                   */
```

```
201 |
202 |                   else
203 |                   {
204 |                         data = ptr->data;
205 |                         (table->table)[val] = ptr->next;
206 |                         free(ptr->key);
207 |                         free(ptr);
208 |                         return data;
209 |                   }
210 |             }
211 |       }
212 |
213 |       /*
214 |       ** If we get here, it means we didn't find the item in the table.
215 |       ** Signal this by returning NULL.
216 |       */
217 |
218 |       return NULL;
219 | }
220 |
221 | /*
222 | ** free_table iterates the table, calling this repeatedly to free
223 | ** each individual node.  This, in turn, calls one or two other
224 | ** functions - one to free the storage used for the key, the other
225 | ** passes a pointer to the data back to a function defined by the user,
226 | ** process the data as needed.
227 | */
228 |
229 | static void free_node(char *key, void *data)
230 | {
231 |       (void) data;
232 |
233 |       if (function)
234 |             function(del(key,the_table));
235 |       else  del(key,the_table);
236 | }
237 |
238 | /*
239 | ** Frees a complete table by iterating over it and freeing each node.
240 | ** the second parameter is the address of a function it will call with a
241 | ** pointer to the data associated with each node.  This function is
242 | ** responsible for freeing the data, or doing whatever is needed with
243 | ** it.
244 | */
245 |
246 | void free_table(hash_table *table, void (*func)(void *))
247 | {
248 |       function = func;
249 |       the_table = table;
250 |
251 |       enumerate( table, free_node);
252 |       free(table->table);
253 |       table->table = NULL;
254 |       table->size = 0;
255 |
256 |       the_table = NULL;
257 |       function = (void (*)(void *))NULL;
258 | }
259 |
260 | /*
261 | ** Simply invokes the function given as the second parameter for each
262 | ** node in the table, passing it the key and the associated data.
263 | */
264 |
265 | void enumerate( hash_table *table, void (*func)(char *, void *))
266 | {
267 |       unsigned i;
268 |       bucket *temp;
269 |
270 |       for (i=0;i<table->size; i++)
271 |       {
272 |             if ((table->table)[i] != NULL)
273 |             {
274 |                   for (temp = (table->table)[i];
275 |                         NULL != temp;
276 |                         temp = temp -> next)
277 |                   {
278 |                         func(temp -> key, temp->data);
279 |                   }
280 |             }
281 |       }
282 | }
283 |
284 |
285 | #ifdef TEST
286 |
287 | #include <stdio.h>
288 |
289 | void printer(char *string, void *data)
290 | {
291 |       printf("%s: %s\n", string, (char *)data);
292 | }
293 |
294 | int main(void)
295 | {
296 |       hash_table table;
297 |
298 |       char *strings[] = {
299 |             "The first string",
300 |             "The second string",
```

```
301 |             "The third string",
302 |             "The fourth string",
303 |             "A much longer string than the rest in this example.",
304 |             "The last string",
305 |             NULL
306 |             };
307 |
308 |       char *junk[] = {
309 |             "The first data",
310 |             "The second data",
311 |             "The third data",
312 |             "The fourth data",
313 |             "The fifth datum",
314 |             "The sixth piece of data"
315 |             };
316 |
317 |       int i;
318 |       void *j;
319 |
320 |       construct_table(&table,200);
321 |
322 |       for (i = 0; NULL != strings[i]; i++ )
323 |             insert(strings[i], junk[i], &table);
324 |
325 |       for (i=0;NULL != strings[i];i++)
326 |       {
327 |             printf("\n");
328 |             enumerate(&table, printer);
329 |             del(strings[i],&table);
330 |       }
331 |
332 |       for (i=0;NULL != strings[i];i++)
333 |       {
334 |             j = lookup(strings[i], &table);
335 |             if (NULL == j)
336 |                   printf("\n'%s' is not in table",strings[i]);
337 |             else  printf("\nERROR: %s was deleted but is still in table.",
338 |                   strings[i]);
339 |       }
340 |       free_table(&table, NULL);
341 |       return 0;
342 | }
343 |
344 | #endif /* TEST */
```

TEXT STATISTICS

9339 characters
344 lines

LEXICAL STATISTICS

18 comments [std-C]
0 comments [C++]
6 preprocessor instructions
0 constants [character]
17 constants [string]
15 constants [numeric]

SYMBOL TABLE

```
NULL            21     22     40     47     89     92     93     96    106    126    146
       147     149    154    166    168    169    179    180    185    218    253    256
       257     272    275    305    322    325    332    335    340
TEST            285
bucket           34     37+    81     91+   125+   144    166    268
construct_table          31    320
data             78     97+    98    111    112+   129+   132    152    165    187+   191
       204+    208    229    231    278    289    291
del             162    234    235    329
enumerate       251    265    328
free            189    190    206    207    252
free_node       229    251
free_table              246    340
func            246    248    265    278
function         21    233    234    248    257
h         3      4    287
hash             57     80    143    164
hash_table               22     31+    78    141    162    246    265    296
i        33      46+    47     60     64     65    267    270+   272    274    317    322+
       323+    325+   329    332+   334    336    338
insert           78    323
j       318     334    335
junk            308    323
key              78     80     95+   107+   128+   141    143    151+   162    164    183+
       189     206    229    234    235    278
last            166    179    181    185    188
lookup          141    334
main            294
malloc           37     91    125
next             96    106    130    149    181    188+   205    276
old_data        109    111    113
printer         289    328
printf          291    327    336    337
ptr              81    106+   107    111    112    125    126    128    129    130    131
       144     149+   151    152    166    179    180    181+   183    187    188    189
       190     204    205    206    207
ret_val          59     65     66     69
size             31     36+    37     42     46     80    143    164    254    270
size_t           31     33
stdio           287
stdlib            4
strcmp          107    151    183
strdup           95    128
string            3     57     62     64     67    289    291
strings         298    322    323    325    329    332    334    336    338
table            31     36     37+    38+    42     43     48     78     80     89+    91+
        92+     95+    96+    97+    98+   106+   130+   131+   141    143    146+   149+
       162     164    168+   179+   205+   246    249    251    252+   253+   254    265
       270     272+   274+   296    320    323    328    329    334    340
temp             34     38     40     47    268    274    275    276+   278+
the_table        22    234    235    249    256
val              80     89     91     92     95     96     97     98    106    130    131
       143     146    149    164    168    179    205
```

```
/* +++Date last modified: 05-Jul-1997 */

#ifndef HASH__H
#define HASH__H

#include <stddef.h>           /* For size_t     */

/*
** A hash table consists of an array of these buckets.  Each bucket
** holds a copy of the key, a pointer to the data associated with the
** key, and a pointer to the next bucket that collided with this one,
** if there was one.
*/

typedef struct bucket {
    char *key;
    void *data;
    struct bucket *next;
} bucket;

/*
** This is what you actually declare an instance of to create a table.
** You then call 'construct_table' with the address of this structure,
** and a guess at the size of the table.  Note that more nodes than this
** can be inserted in the table, but performance degrades as this
** happens.  Performance should still be quite adequate until 2 or 3
** times as many nodes have been inserted as the table was created with.
*/

typedef struct hash_table {
    size_t size;
    bucket **table;
} hash_table;

/*
** This is used to construct the table.  If it doesn't succeed, it sets
** the table's size to 0, and the pointer to the table to NULL.
*/

hash_table *construct_table(hash_table *table,size_t size);

/*
** Inserts a pointer to 'data' in the table, with a copy of 'key' as its
** key.  Note that this makes a copy of the key, but NOT of the
** associated data.
*/

void *insert(char *key,void *data,struct hash_table *table);

/*
** Returns a pointer to the data associated with a key.  If the key has
** not been inserted in the table, returns NULL.
*/

void *lookup(char *key,struct hash_table *table);

/*
** Deletes an entry from the table.  Returns a pointer to the data that
** was associated with the key so the calling code can dispose of it
** properly.
*/

void *del(char *key,struct hash_table *table);

/*
** Goes through a hash table and calls the function passed to it
** for each node that has been inserted.  The function is passed
** a pointer to the key, and a pointer to the data associated
** with it.
*/

void enumerate(struct hash_table *table,void (*func)(char *,void *));

/*
** Frees a hash table.  For each node that was inserted in the table,
** it calls the function whose address it was passed, with a pointer
** to the data that was in the table.  The function is expected to
** free the data.  Typical usage would be:
** free_table(&table, free);
** if the data placed in the table was dynamically allocated, or:
** free_table(&table, NULL);
** if not.  ( If the parameter passed is NULL, it knows not to call
** any function with the data. )
*/

void free_table(hash_table *table, void (*func)(void *));

#endif /* HASH__H */
```

```
TEXT STATISTICS

  2713 characters
    88 lines

LEXICAL STATISTICS

    11 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    HASH__H          3       4
    bucket          15      18      19      32
    construct_table         40
    data            17      48
    del             63
    enumerate       72
    free_table              86
    func            72      86
    h        6
    hash_table              30      33      40+     48      55      63      72      86
    insert          48
    key             16      48      55      63
    lookup          55
    next            18
    size            31      40
    size_t          31      40
    stddef           6
    table           32      40      48      55      63      72      86
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #include <stdio.h>
  4 | #include <stdlib.h>
  5 | #include <string.h>
  6 |
  7 | #define NUL '\000'
  8 | #define BEL '\007'
  9 | #define LINE_LEN 132
 10 |
 11 | void give_up(char *msg)
 12 | {
 13 |       putchar(BEL);
 14 |       puts(msg);
 15 |       exit(-1);
 16 | }
 17 |
 18 | int main(int argc, char *argv[])
 19 | {
 20 |       FILE *infile;
 21 |       char line[LINE_LEN + 2];                  /* Allow for '\n' & NUL */
 22 |       int i, N = 0;
 23 |
 24 |       if (2 > argc)
 25 |            give_up("Usage: HEAD file [number_of_lines]");
 26 |       if (NULL == (infile = fopen(argv[1], "r")))
 27 |            give_up("Unable to open input file");
 28 |       if (2 < argc)
 29 |            N = atoi(argv[2]);
 30 |       if (!N) N = 4;
 31 |       for (i = 0; i < N; ++i)
 32 |       {
 33 |            if (NULL == fgets(line, LINE_LEN + 1, infile))
 34 |                 break;
 35 |            line[LINE_LEN + 1] = NUL;           /* Allow too-long lines */
 36 |            fputs(line, stdout);
 37 |            if (!strrchr(line, '\n'))
 38 |                 i -= 1;                        /* More to read         */
 39 |       }
 40 |       fclose(infile);
 41 |       return EXIT_SUCCESS;
 42 | }
```

TEXT STATISTICS

```
  1090 characters
    42 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     3 constants [character]
     3 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    BEL              8      13
    EXIT_SUCCESS             41
    FILE            20
    LINE_LEN         9      21      33      35
    N       22      29      30+     31
    NUL              7      35
    NULL            26      33
    argc            18      24      28
    argv            18      26      29
    atoi            29
    exit            15
    fclose          40
    fgets           33
    fopen           26
    fputs           36
    give_up         11      25      27
    h        3       4       5
    i       22      31+     38
    infile          20      26      33      40
    line            21      33      35      36      37
    main            18
    msg             11      14
    putchar         13
    puts            14
    stdio            3
    stdlib           4
    stdout          36
    string           5
    strrchr         37
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  HEXDUMP.C - Dump a file.
**
**  Originally written By Paul Edwards
**  Released to the public domain
**
**  Modified for SNIPPETS by Bob Stout
**
**  Uses ERR_EXIT.C and FERROF.C, also in SNIPPETS
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "errors.h"

static void dodump(FILE *fp, long start, long count);

main(int argc, char **argv)
{
      FILE *fp;
      long start, count, length;

      if (argc < 2)
            ErrExit("Usage: HEXDUMP file_name [start] [length]");
      if (argc > 2)
            sscanf(argv[2], "%li", &start);
      else  start = 0L;
      if (argc > 3)
            sscanf(argv[3], "%li", &count);
      else  count = -1L;

      fp = cant(argv[1], "rb");

      fseek(fp, 0L, SEEK_END);
      length = ftell(fp);
      if (start > length)
      {
            ErrExit("Can't find position %ld in a %ld byte file",
                  start, length);
      }
      if (fseek(fp, start, SEEK_SET))
            ErrExit("Unable to find position %ld", start);

      dodump(fp, start, count);
      return (EXIT_SUCCESS);
}

static void dodump(FILE *fp, long start, long count)
{
      int c, pos1, pos2, posn = (int)(start % 16L);
      long x = 0L;
      char prtln[80];

      while (((c = fgetc(fp)) != EOF) && (x != count))
      {
            if (0 == (posn % 16) || 0 == x)
            {
                  memset(prtln,' ',sizeof prtln);
                  sprintf(prtln,"%0.6X:", start + x);
                  prtln[7] = ' ';
                  pos1 = 8 + (int)(3 * posn);
                  if (posn > 3)
                        ++pos1;
                  if (posn > 7)
                        ++pos1;
                  if (posn > 11)
                        ++pos1;
                  pos2 = 60 + (int)(posn);
            }
            sprintf(prtln + pos1, "%0.2X ", c);
            if (isprint(c))
                  sprintf(prtln + pos2, "%c", c);
            else  sprintf(prtln + pos2, ".");
            pos1 += 3;
            *(prtln+pos1) = ' ';
            pos2++;
            if (posn % 4 == 3)
                  *(prtln + pos1++) = ' ';
            if (posn % 16 == 15)
            {
                  printf("%s\n", prtln);
                  posn = 0;
            }
            else  ++posn;
            ++x;
      }
      if (posn % 16)
            printf("%s\n", prtln);
      return;
}
```

TEXT STATISTICS

```
  2515 characters
    94 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     4 constants [character]
    13 constants [string]
    29 constants [numeric]
```

SYMBOL TABLE

```
    EOF             58
    EXIT_SUCCESS            49
    ErrExit         28      42      46
    FILE            20      24      52
    SEEK_END        38
    SEEK_SET        45
    argc            22      27      29      32
    argv            22      30      33      36
    c       54      58      74      75      76
    cant            36
    count           20      25      33      34      48      52      58
    ctype           17
    dodump          20      48      52
    fgetc           58
    fp              20      24      36      38      39      45      48      52      58
    fseek           38      45
    ftell           39
    h       14      15      16      17
    isprint         75
    length          25      39      40      43
    main            22
    memset          62
    pos1            54      65      67      69      71      74      78      79      82
    pos2            54      72      76      77      80
    posn            54      60      65      66      68      70      72      81      83      86      88
            91
    printf          85      92
    prtln           56      62+     63      64      74      76      77      79      82      85      92
    sprintf         63      74      76      77
    sscanf          30      33
    start           20      25      30      31      40      43      45      46      48      52      54
            63
    stdio           14
    stdlib          15
    string          16
    x       55      58      60      63      89
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  HEXORINT.C - Detect if a string denotes a hex or decimal
**  number by detecting a leading "0X" or trailing "H" string.
**
**  public domain demo by Bob Stout
*/

#include <stdlib.h>
#include <string.h>
#include "sniptype.h"
#include "numcnvrt.h"

/*
**  Let strtol() do most of the work
*/

long hexorint(const char *string)
{
      int radix = 0;
      char *dummy, valstr[128];

      strcpy(valstr, string);
      if (strchr("Hh", LAST_CHAR(valstr)))
      {
            LAST_CHAR(valstr) = NUL;
            radix = 16;
      }
      return strtol(valstr, &dummy, radix);
}

/*
**  Test code follows - compile with TEST macro defined to test
*/

#ifdef TEST

#include <stdio.h>

main(int argc, char *argv[])
{
      long val;

      while (--argc)
      {
            val = hexorint(*(++argv));
            printf("Value of %s = %ld = %#lx\n", *argv, val, val);
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1024 characters
    53 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     4 constants [string]
     3 constants [numeric]

SYMBOL TABLE

    EXIT_SUCCESS            50
    LAST_CHAR       25      27
    NUL             27
    TEST            37
    argc            41      45
    argv            41      47      48
    dummy           22      30
    h       10      11      39
    hexorint        19      47
    main            41
    printf          48
    radix           21      28      30
    stdio           39
    stdlib          10
    strchr          25
    strcpy          24
    string          11      19      24
    strtol          30
    val             43      47      48+
    valstr          22      24      25      27      30
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | #ifndef HILOBYTE__H
   4 | #define HILOBYTE__H
   5 |
   6 | #define LoByte(x) ((unsigned char)((x) & 0xff))
   7 | #define HiByte(x) ((unsigned char)((unsigned short)(x) >> 8))
   8 |
   9 | #endif /* HILOBYTE__H */
```

TEXT STATISTICS

```
   228 characters
     9 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    HILOBYTE__H              3       4
    HiByte           7
    LoByte           6
    x        6+      7+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #define _ putchar
  4 | #define __ ^
  5 | #define ___ /
  6 | #include <stdio.h>
  7 | void main()
  8 | {_((2*2*2*2*2*2*3*41)___
  9 | 64__(3*17));_((2*2*2*13*89)___
 10 | 104__(2*2*3*5));_((2*2*2*23*61)___
 11 | 92__(2*11));_((2*2*2*2*2*3*3)___
 12 | 4__(2*2*3*3));_((2*2*2*2*7*13)___
 13 | 26__(3*29));_((2*2*3*7*19)___
 14 | 28__(3*7));_((2*2*2*2*13*29)___
 15 | 104__(2*13));_((2*2*2*2*2*2*3*19)___
 16 | 114__(3*29));_((2*2*5*107)___
 17 | 20__(2*2));_((3*11*97)___
 18 | 97__(83));_((2*2*2*2*2*3*11)___
 19 | 11__(2*2*3));_((2*5*11*13)___
 20 | 55__(2*3*3*7));_((2*3*7*83)___
 21 | 83__(2*2));_((79)___
 22 | 79__(11));
 23 | }
```

TEXT STATISTICS

```
   581 characters
    23 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
   120 constants [numeric]
```

SYMBOL TABLE

```
    _        3       8       9      10      11      12      13      14      15      16      17      18
            19      20      21
    __               4
    ___              5       8       9      10      11      12      13      14      15      16      17
            18      19      20      21
    h        6
    main             7
    putchar          3
    stdio            6
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Originally published as part of the MicroFirm Function Library
**
**  Copyright 1986, S.E. Margison
**  Copyright 1989, Robert B.Stout
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Make an ascii hexadecimal string into an integer.
*/

#include <stdio.h>
#include <ctype.h>
#include "numcnvrt.h"

unsigned int hstr_i(char *cptr)
{
      unsigned int i, j = 0;

      while (cptr && *cptr && isxdigit(*cptr))
      {
            i = *cptr++ - '0';
            if (9 < i)
                  i -= 7;
            j <<= 4;
            j |= (i & 0x0f);
      }
      return(j);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      char *arg;
      unsigned int x;

      while (--argc)
      {
            x = hstr_i(arg = *++argv);
            printf("Hex %s = %d\n", arg, x, x);
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1135 characters
    53 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     1 constants [character]
     2 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

EXIT_SUCCESS		50			
TEST		35			
arg		42	47	48	
argc		40	45		
argv		40	47		
cptr		20	24+	26	
ctype		17			
h	16	17	37	38	
hstr_i		20	47		
i	22	26	27	28	30
isxdigit		24			
j	22	29	30	32	
main		40			
printf		48			
stdio		16	37		
stdlib		38			
x	43	47	48+		

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  HUGEREAD.C - "Universal" PC read and write functions using huge data
**               and far pointers.
**
**  NOTES:
**
**  1. If these functions are called with a prototype in scope, passed
**     parameters will be coerced to the proper data types.
**
**  2. Since these call read() and write(), all normal mode flags which
**     are supported by individual compilers will be honored.
**
**  3. In small data memory models (S, T, and M), an intermediate buffer
**     is allocated and used. In large data models (L and C), the data
**     are read/written directly from/to target memory.
**
**  4. Like many mixed-model functions, this may generate lots of warnings
**     with many compilers. Despite this, it really does generate correct
**     code for all major PC compilers.
**
**  Original Copyright 1992 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <dos.h>
#include <io.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include "extkword.h"
#include "minmax.h"
#include "snpdskio.h"
#include "snpdosys.h"

/*
**  Get the largest buffer possible.
*/

static size_t gettmp(char **buf)
{
      size_t bufsiz;

      for (bufsiz = 0x4000; bufsiz >= 128; bufsiz >>= 1)
      {
            if (NULL != (*buf = (char *) malloc(bufsiz)))
                  return bufsiz;
      }
      return 0;
}

/*
**  Read any size block to anywhere in memory
*/

long hugeread(int fh, unsigned char FAR *buf, long size)
{
      long count;
      size_t bufsiz;
      char *tmp;
      long ercode = size;

      if (4 > sizeof(void *))
      {
            if (0 == (bufsiz = gettmp(&tmp)))
                  return -1L;
      }
      else
      {
            tmp = (char *)buf;
            bufsiz = 0x4000;
      }

      buf = farnormal(buf);
      while (0 < (count = min(size, (long)bufsiz)))
      {
            int i, numread = read(fh, tmp, (size_t)count);

            if (1 > numread || numread != (int)count)
                  return -1L;
            if (4 > sizeof(void *))
            {
                  for (i = 0; i < count; ++i)
                        buf[i] = tmp[i];
            }
            buf = farnormal(buf + count);
            size -= count;
            if (2 < sizeof(void *))
                  tmp = (char *)buf;
      }
      return ercode;
}

/*
**  Write any size block from anywhere in memory
```

```
101 | */
102 |
103 | long hugewrite(int fh, unsigned char FAR *buf, long size)
104 | {
105 |       long count;
106 |       size_t bufsiz;
107 |       char *tmp;
108 |       long ercode = size;
109 |
110 |       if (4 > sizeof(void *))
111 |       {
112 |             if (0 == (bufsiz = gettmp(&tmp)))
113 |                   return -1L;
114 |       }
115 |       else
116 |       {
117 |             tmp = (char *)buf;
118 |             bufsiz = 0x4000;
119 |       }
120 |
121 |       buf = farnormal(buf);
122 |       while (0 < (count = min(size, (long)bufsiz)))
123 |       {
124 |             int i, numwrite;
125 |
126 |             if (4 > sizeof(void *))
127 |             {
128 |                   for (i = 0; i < count; ++i)
129 |                         tmp[i] = buf[i];
130 |             }
131 |             numwrite = write(fh, tmp, (size_t)count);
132 |             if (1 > numwrite || numwrite != (int)count)
133 |                   return -1L;
134 |             buf = farnormal(buf + count);
135 |             size -= count;
136 |             if (2 < sizeof(void *))
137 |                   tmp = (char *)buf;
138 |       }
139 |       return ercode;
140 | }
141 |
142 | /*
143 | **  Read any size block to anywhere in memory
144 | */
145 |
146 | long hugefread(FILE *fp, char FAR *buf, long size)
147 | {
148 |       long count;
149 |       size_t bufsiz;
150 |       char *tmp;
151 |       long ercode = size;
152 |
153 |       if (4 > sizeof(void *))
154 |       {
155 |             if (0 == (bufsiz = gettmp(&tmp)))
156 |                   return -1L;
157 |       }
158 |       else
159 |       {
160 |             tmp = (char *)buf;
161 |             bufsiz = 0x4000;
162 |       }
163 |
164 |       buf = farnormal(buf);
165 |       while (0 < (count = min(size, (long)bufsiz)))
166 |       {
167 |             int i, numread = fread(tmp, 1, (size_t)count, fp);
168 |
169 |             if (1 > numread || numread != (int)count)
170 |                   return -1L;
171 |             if (4 > sizeof(void *))
172 |             {
173 |                   for (i = 0; i < count; ++i)
174 |                         buf[i] = tmp[i];
175 |             }
176 |             buf = farnormal(buf + count);
177 |             size -= count;
178 |             if (2 < sizeof(void *))
179 |                   tmp = (char *)buf;
180 |       }
181 |       return ercode;
182 | }
183 |
184 | /*
185 | **  Write any size block from anywhere in memory
186 | */
187 |
188 | long hugefwrite(FILE *fp, char FAR *buf, long size)
189 | {
190 |       long count;
191 |       size_t bufsiz;
192 |       char *tmp;
193 |       long ercode = size;
194 |
195 |       if (4 > sizeof(void *))
196 |       {
197 |             if (0 == (bufsiz = gettmp(&tmp)))
198 |                   return -1L;
199 |       }
200 |       else
```

```
201 |       {
202 |             tmp = (char *)buf;
203 |             bufsiz = 0x4000;
204 |       }
205 |
206 |       buf = farnormal(buf);
207 |       while (0 < (count = min(size, (long)bufsiz)))
208 |       {
209 |             int i, numwrite;
210 |
211 |             if (4 > sizeof(void *))
212 |             {
213 |                   for (i = 0; i < count; ++i)
214 |                         tmp[i] = buf[i];
215 |             }
216 |             numwrite = fwrite(tmp, 1, (size_t)count, fp);
217 |             if (1 > numwrite || numwrite != (int)count)
218 |                   return -1L;
219 |             buf = farnormal(buf + count);
220 |             size -= count;
221 |             if (2 < sizeof(void *))
222 |                   tmp = (char *)buf;
223 |       }
224 |       return ercode;
225 | }
```

TEXT STATISTICS

```
  5723 characters
   225 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     4 constants [string]
    46 constants [numeric]
```

SYMBOL TABLE

```
    FAR             61     103     146     188
    FILE           146     188
    NULL            51
    buf             45      51      61      75      79+     89      91+     94     103     117     121+
           129     134+    137     146     160     164+    174     176+    179     188     202     206+
           214     219+    222
    bufsiz          47      49+     51      52      64      70      76      80     106     112     118
           122     149     155     161     165     191     197     203     207
    count           63      80      82      84      88      91      92     105     122     128     131
           132     134     135     148     165     167     169     173     176     177     190     207
           213     216     217     219     220
    dos             31
    ercode          66      96     108     139     151     181     193     224
    farnormal       79      91     121     134     164     176     206     219
    fh              61      82     103     131
    fp             146     167     188     216
    fread          167
    fwrite         216
    gettmp          45      70     112     155     197
    h       31      32      33      34      35
    hugefread      146
    hugefwrite             188
    hugeread        61
    hugewrite      103
    i       82      88+     89+    124     128+    129+    167     173+    174+    209     213+    214+
    io              32
    malloc          51
    min             80     122     165     207
    numread         82      84+    167     169+
    numwrite       124     131     132+    209     216     217+
    read            82
    size            61      66      80      92     103     108     122     135     146     151     165
           177     188     193     207     220
    size_t          45      47      64      82     106     131     149     167     191     216
    stddef          34
    stdio           35
    stdlib          33
    tmp             65      70      75      82      89      94     107     112     117     129     131
           137     150     155     160     167     174     179     192     197     202     214     216
           222
    write          131
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  hugesort.c -- huge qsort() -- public domain by Raymond Gardner 6/92
  5 | **  tested with Borland C++ 3.0 (C mode)
  6 | */
  7 |
  8 | #include "big_mall.h"
  9 | #include "snipsort.h"
 10 |
 11 | static void swap(char HUGE *a, char HUGE *b, unsigned n)
 12 | {
 13 |       char tmp;
 14 |
 15 |       do
 16 |       {
 17 |             tmp = *a; *a++ = *b; *b++ = tmp;
 18 |       } while ( --n );
 19 | }
 20 |
 21 | void hugesort(void HUGE *basep,
 22 |               unsigned   nel,
 23 |               unsigned   width,
 24 |               int      (*comp)(void HUGE *, void HUGE *))
 25 | {
 26 |       char HUGE *i, HUGE *j;
 27 |       unsigned int lnel, rnel;
 28 |       char HUGE *base = (char HUGE *)basep;
 29 |
 30 |       while (nel > 1)
 31 |       {
 32 |             swap(base, base + (long)width * (nel / 2), width);
 33 |             for (i = base, j = base + (long)width * nel; ; )
 34 |             {
 35 |                   do
 36 |                         j -= width;
 37 |                   while ( (*comp)(j, base) > 0 );
 38 |                   do
 39 |                         i += width;
 40 |                   while ( i < j && (*comp)(i, base) < 0 );
 41 |                   if (i >= j)
 42 |                         break;
 43 |                   swap(i, j, width);
 44 |             }
 45 |             swap(j, base, width);
 46 |             lnel = (unsigned)((long)(j - base) / width);
 47 |             rnel = nel - lnel - 1;
 48 |             j += width;
 49 |             if (lnel < rnel)
 50 |             {
 51 |                   hugesort(base, lnel, width, comp);
 52 |                   base = j;
 53 |                   nel = rnel;
 54 |             }
 55 |             else
 56 |             {
 57 |                   hugesort(j, rnel, width, comp);
 58 |                   nel = lnel;
 59 |             }
 60 |       }
 61 | }
 62 |
 63 | #ifdef TEST
 64 |
 65 | #include <stdio.h>
 66 | #include <stdlib.h>
 67 | #include "errors.h"
 68 |
 69 | #define PADSIZE 300
 70 |
 71 | typedef struct x {
 72 |     int key;
 73 |     char pad[PADSIZE];
 74 |     } X;
 75 |
 76 | int cmpv(void HUGE *a, void HUGE *b) /* (note void HUGE *) passed here */
 77 | {
 78 |       return ((X HUGE *)a)->key < ((X HUGE *)b)->key ? -1 :
 79 |             ((X HUGE *)a)->key > ((X HUGE *)b)->key ? 1 : 0;
 80 | }
 81 |
 82 | int main(int argc, char **argv)
 83 | {
 84 |       X HUGE *v;
 85 |       int n;
 86 |       int i, j;
 87 |
 88 |       n = 300;                            /* default element count */
 89 |       if (argc > 1)
 90 |             n = atoi(argv[1]);
 91 |       printf("test with %d elements\n", n);
 92 |
 93 |       if (NULL == (v = BigMalloc(sizeof(X), n)))
 94 |             ErrExit("Insufficient memory"); /* be sure we got memory */
 95 |
 96 |       for (i = 0; i < n; ++i)             /* random init */
 97 |       {
 98 |             v[i].key = rand();
 99 |             for (j = 0; j < PADSIZE; ++j)
100 |                   v[i].pad[j] = rand();
```

```
 101 |       }
 102 |       for (i = 0; i < n; ++i)             /* display before */
 103 |             printf(" %d", v[i].key);
 104 |       printf("\n");
 105 |       hugesort(v, n, sizeof(X), cmpv);    /* sort it */
 106 |       for ( i = 0; i < n; ++i )           /* display after */
 107 |             printf(" %d", v[i].key);
 108 |       printf("\n");
 109 |       BigFree(v);
 110 |       return 0;
 111 | }
 112 |
 113 | #endif /* TEST */
```

TEXT STATISTICS

```
  2967 characters
   113 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     9 constants [string]
    17 constants [numeric]
```

SYMBOL TABLE

```
    BigFree        109
    BigMalloc       93
    ErrExit         94
    HUGE            11+     21      24+     26+     28+     76+     78+     79+     84
    NULL            93
    PADSIZE         69      73      99
    TEST            63
    X       74      78+     79+     84      93      105
    a       11      17+     76      78      79
    argc            82      89
    argv            82      90
    atoi            90
    b       11      17+     76      78      79
    base            28      32+     33+     37      40      45      46      51      52
    basep           21      28
    cmpv            76      105
    comp            24      37      40      51      57
    h       65      66
    hugesort        21      51      57      105
    i       26      33      39      40+     41      43      86      96+     98      100     102+    103
            106+    107
    j       26      33      36      37      40      41      43      45      46      48      52      57
            86      99+     100
    key             72      78+     79+     98      103     107
    lnel            27      46      47      49      51      58
    main            82
    n       11      18      85      88      90      91      93      96      102     105     106
    nel             22      30      32      33      47      53      58
    pad             73      100
    printf          91      103     104     107     108
    rand            98      100
    rnel            27      47      49      53      57
    stdio           65
    stdlib          66
    swap            11      32      43      45
    tmp             13      17+
    v       84      93      98      100     103     105     107     109
    width           23      32+     33      36      39      43      45      46      48      51      57
    x       71
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** ifactor.c -- print prime factorization of a number
**
** Ray Gardner -- 1985 -- public domain
*/

#include <stdio.h>
#include <stdlib.h>

int prevfact = 0;
void factor (long);
void show (long, int);

main (int argc, char *argv[])
{
      while ( --argc )
            factor(atol(*++argv));
      return 0;
}

void factor (long n)
{
      long d;
      int k;
      long n0 = n;
      prevfact = 0;

      printf("%ld  ",n);
      if ( n < 2 )
      {
            printf("is less than 2.\n");
            return;
      }
      else if ( n > 2 )
      {
            d = 2;
            for ( k = 0; n % d == 0; k++ )
                  n /= d;
            if ( k )
                  show(d,k);
            for ( d = 3; d * d <= n; d += 2 )
            {
                  for ( k = 0; n % d == 0; k++ )
                        n /= d;
                  if ( k )
                        show(d,k);
            }
      }
      if ( n > 1 )
      {
            if ( n == n0 )
                  printf(" is prime");
            else  show(n,1);
      }
      printf("\n");
}

void show (long d, int k)
{
      if ( prevfact )
            printf(" * ");
      else  printf(" = ");
      prevfact++;
      printf("%ld",d);
      if ( k > 1 )
            printf("^%d",k);
}
```

TEXT STATISTICS

```
  1381 characters
    69 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     8 constants [string]
    15 constants [numeric]
```

SYMBOL TABLE

```
    argc            16      18
    argv            16      19
    atol            19
    d       25      38      39      40      42      43+     45      46      48      60      66
    factor          13      19      23
    h        9      10
    k       26      39+     41      42      45+     47      48      60      67      68
    main            16
    n       23      27      30      31      36      39      40      43      45      46      51      53
            55
    n0              27      53
    prevfact        12      28      62      65
    printf          30      33      54      57      63      64      66      68
    show            14      42      48      55      60
    stdio            9
    stdlib          10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Convert English measurement units
**
**  Takes command line arguments in inches and converts to:
**
**  1. feet and inches (expressed as floating point)
**  2. feet and inches (expressed as fraction)
**
**  public domain demo by Bob Stout
**  uses ROUND.H from SNIPPETS
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "round.h"

#define BASE 64.0

void cnvrt_inches(double  input,
                  double *feet,
                  double *inches,
                  double *dec_inches,
                  double *num_inches,
                  double *den_inches)
{
      /*
      **  Split feet and inches
      */

      *feet   = floor(input / 12.0);
      *inches = fmod(input, 12.0);

      /*
      **  Get integer inches and fractions
      */

      *num_inches = modf(*inches, dec_inches) * BASE;

      *num_inches = fround(*num_inches, 0);
      if (0.0 == *num_inches)
            return;

      /*
      **  Reduce fractions to lowest common denominator
      */

      for (*den_inches = BASE;
            0.0 == fmod(*num_inches, 2.0);
            *den_inches /= 2.0, *num_inches /= 2.0)
      {
            ;
      }
}

main(int argc, char *argv[])
{
      double arg, feet, inches, dec, num, den, dummy;

      while (--argc)
      {
            arg = atof(*(++argv));
            cnvrt_inches(arg, &feet, &inches, &dec, &num, &den);
            printf("%f Inches = %d' %.5f\" or %d' %d",
                  arg, (int)feet, inches, (int)feet, (int)dec);
            if (0.0 == num)
                  puts("\"");
            else
            {
                  printf("-%d/%d\"", (int)num, (int)den);
                  if (modf(num, &dummy))
                        puts(" (approx.)");
                  else  puts("");
            }
      }
      return EXIT_SUCCESS;
}
```

TEXT STATISTICS

```
  1934 characters
    79 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     6 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    BASE            20      40      50
    EXIT_SUCCESS            78
    arg             60      64      65      67
    argc            58      62
    argv            58      64
    atof            64
    cnvrt_inches            22      65
    dec             60      65      67
    dec_inches              25      40
    den             60      65      72
    den_inches              27      50      52
    dummy           60      73
    feet            23      33      60      65      67+
    floor           33
    fmod            34      51
    fround          42
    h       15      16      17
    inches          24      34      40      60      65      67
    input           22      33      34
    main            58
    math            17
    modf            40      73
    num             60      65      68      72      73
    num_inches              26      40      42+     43      51      52
    printf          66      72
    puts            69      74      75
    stdio           15
    stdlib          16
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  INDEX.C
**
**  This is the indexing function for creating a binary file from an ASCII
**  file formatted as follows:
**
**  Mark Corgan
**  550 Foothill Rd.
**  Gardnerville, NV 89410
**  (702) 265-2388
**  .
**  Hello World
**  123 Anywhere St.
**  Anytown, CA 12345
**  (123) 456-7890
**  .
**  etc...
**
**  The period is what the companion LOOKUP.C looks for to indicate the end
**  of record. Of course, you could have any format you like, so long as the
**  first line is the information you are looking for. Also, there is no
**  limit to the number of lines of infomation after the first line and
**  before the period as fputs() continues until the period. Enjoy!
**
**  by Mark Corgan, 09-May-1993, and donated to the public domain
*/

#include <stdio.h>
#include <stdlib.h>
#include "errors.h"                 /* for cant()                        */
#include "indxlook.h"               /* definitions for INDEX and LOOKUP */

struct tree_node                    /* node in tree */
{
   struct tree_node *l_ptr, *r_ptr;   /* left and right pointers */
   INDEX *data_ptr;                   /* data pointer */
};

typedef struct tree_node TREE_NODE;

void        write_index(char *txtfile, char *ndxfile);
void        save_tree(TREE_NODE * root, FILE *fp);
TREE_NODE  *make_tree(FILE *fp, long *cnt_ptr);
TREE_NODE  *insert_tree(TREE_NODE *root, INDEX *rec_ptr, long *cnt_ptr);
long        bsearch_(FILE *ifp, long first, long last, char *target);

int main(int argc, char *argv[])
{
      if (argc != 3)
      {
            puts("Usage: INDEX text_file_name index_file_name\n");
            puts("Note: The text file must consist of a number of records "
                 "separated by lines");
            puts("      containing a single period (\".\")");
            return EXIT_FAILURE;
      }
      write_index(argv[1], argv[2]);
      return EXIT_SUCCESS;
}

void write_index(char *txtfile, char *ndxfile)
{
      FILE *afp, *ifp;                    /* types of files       */
      TREE_NODE *root;                    /* index tree           */
      static INDEX header = {"!", 0};     /* dummy header node    */

      afp = cant(txtfile, "r");
      if ((root = make_tree(afp, &header.pos)) != NULL)
      {
            ifp = cant(ndxfile, "wb");
            fwrite((char *) &header, sizeof(header), 1, ifp);
            save_tree(root, ifp);
            fclose(ifp);
            printf("\n%ld records\n", header.pos);
      }
      fclose(afp);
}

/*
** Make index tree
*/

TREE_NODE *make_tree(FILE *fp,            /* file                 */
                     long *cnt_ptr)       /* count of records     */
{
      TREE_NODE *root = NULL, *temp_ptr;  /* add node to tree     */
      char line[MAX_LINE];                /* next line            */
      long start_pos = 0;
      INDEX *next_ptr;                    /* next key, pos pair   */
      /* starting with new record */
      Boolean_T new_record = True_, have_mem = True_;

      *cnt_ptr = 0;
      while (start_pos = ftell(fp), have_mem && fgets(line,sizeof(line), fp))
      {
            if (new_record)
            {
                  if ((next_ptr = (INDEX *) malloc(sizeof(INDEX))) != NULL)
```

```
101 |                   {
102 |                         strncpy(next_ptr->key, line, MAX_KEY);
103 |
104 |                         next_ptr->pos = start_pos;
105 |                         temp_ptr      = insert_tree(root, next_ptr, cnt_ptr);
106 |
107 |                         if (temp_ptr)
108 |                               root = temp_ptr;
109 |                   }
110 |                   have_mem = next_ptr && temp_ptr;
111 |             }
112 |             new_record = strcmp(line, END_REC) == 0;
113 |       }
114 |       if (!have_mem)
115 |             fprintf(stderr, "Out of memory. Key: %s\n", line);
116 |
117 |       return root;
118 | }
119 |
120 | /*
121 | ** Save the index tree to a file
122 | */
123 |
124 | void save_tree(TREE_NODE *root, FILE *fp)
125 | {
126 |       if (root)
127 |       {
128 |             save_tree(root->l_ptr, fp);
129 |             fwrite(root->data_ptr, sizeof(INDEX), 1, fp);
130 |             save_tree(root->r_ptr, fp);
131 |       }
132 | }
133 |
134 | /*
135 | ** Add record to tree
136 | */
137 |
138 | TREE_NODE *insert_tree(TREE_NODE *root,         /* pointer to tree      */
139 |                        INDEX *rec_ptr,          /* record to install    */
140 |                        long *cnt_ptr)           /* nodes in tree        */
141 |
142 | {
143 |       if (root)
144 |       {
145 |             int cmp = strncmp(rec_ptr->key, root->data_ptr->key, MAX_KEY);
146 |
147 |             if (cmp < 0)                        /* left side?           */
148 |                   root->l_ptr = insert_tree(root->l_ptr, rec_ptr, cnt_ptr);
149 |             else if (cmp > 0)                   /* right side           */
150 |                   root->r_ptr = insert_tree(root->r_ptr, rec_ptr, cnt_ptr);
151 |             else  fprintf(stderr, "Duplicate key: %s\n", rec_ptr->key);
152 |       }
153 |       else if (root = (TREE_NODE *) malloc(sizeof(TREE_NODE)), root)
154 |       {
155 |             root->data_ptr = rec_ptr;
156 |             root->l_ptr = root->r_ptr = NULL;
157 |             (*cnt_ptr)++;
158 |       }
159 |       return root;
160 | }
161 |
162 | long bsearch_(FILE *ifp, long first, long last, char *target)
163 | {
164 |       long pos, mid =(first + last) / 2;
165 |       INDEX next;
166 |       int cmp;
167 |
168 |       if (mid < first || fseek(ifp, mid * sizeof(INDEX), 0) != 0 ||
169 |           fread((char *) &next, sizeof(INDEX), 1, ifp) == 0)
170 |       {
171 |             pos = -1;
172 |       }
173 |       else  pos = ((cmp = strncmp(target, next.key, MAX_KEY)) == 0) ?
174 |                   next.pos :
175 |                   ((cmp < 0) ? bsearch_(ifp, first, mid - 1, target)
176 |                              : bsearch_(ifp, mid + 1, last, target));
177 |       return pos;
178 | }
```

TEXT STATISTICS

```
  5747 characters
   178 lines
```

LEXICAL STATISTICS

```
    24 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
    12 constants [string]
    21 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       93
    END_REC        112
    EXIT_FAILURE            57
    EXIT_SUCCESS            60
    FILE            44      45      47      65      85     124     162
    INDEX           38      46      67      91    100+    129     139     165     168     169
    MAX_KEY        102     145     173
    MAX_LINE        89
    NULL            70      88     100     156
    TREE_NODE       41      44      45     46+      66      85      88     124    138+    153+
    True_          93+
    afp             65      69      70      78
    argc            49      51
    argv            49     59+
    bsearch_        47     162     175     176
    cant            69      72
    cmp            145     147     149     166     173     175
    cnt_ptr         45      46      86      95     105     140     148     150     157
    data_ptr        38     129     145     155
    fclose          75      78
    fgets           96
    first           47     162     164     168     175
    fp              44      45      85     96+     124     128     129     130
    fprintf        115     151
    fread          169
    fseek          168
    ftell           96
    fwrite          73     129
    h       30      31
    have_mem        93      96     110     114
    header          67      70     73+      76
    ifp             47      65      72      73      74      75     162     168     169     175     176
    insert_tree             46     105     138     148     150
    key            102    145+     151     173
    l_ptr           37     128    148+     156
    last            47     162     164     176
    line            89     96+     102     112     115
    main            49
    make_tree       45      70      85
    malloc         100     153
    mid            164    168+     175     176
    ndxfile         43      63      72
    new_record              93      98     112
    next           165     169     173     174
    next_ptr        91     100     102     104     105     110
    pos             70      76     104     164     171     173     174     177
    printf          76
    puts            53      54      56
    r_ptr           37     130    150+     156
    rec_ptr         46     139     145     148     150     151     155
    root            44      46      66      70      74      88     105     108     117     124     126
           128     129     130     138     143     145    148+    150+    153+    155    156+    159
    save_tree       44      74     124     128     130
    start_pos       90      96     104
    stderr         115     151
    stdio           30
    stdlib          31
    strcmp         112
    strncmp        145     173
    strncpy        102
    target          47     162     173     175     176
    temp_ptr        88     105     107     108     110
    tree_node       35      37      41
    txtfile         43      63      69
    write_index             43      59      63
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  INDXLOOK.H
  5 | **
  6 | **  SNIPPETS header file for INDEX.C and LOOKUP.C
  7 | **
  8 | **  public domain by Mark Corgan
  9 | */
 10 |
 11 | #include "sniptype.h"           /* for True_, False_, Boolean_T */
 12 | #define MAX_LINE           40       /* maximum line length         */
 13 | #define MAX_KEY            26       /* maximum key size         */
 14 | #define END_REC         ".\n"       /* end of a record          */
 15 |
 16 | struct index_rec                    /* index record             */
 17 | {
 18 |     char key[MAX_KEY];            /* name                   */
 19 |     long pos;                     /* position                 */
 20 | };
 21 |
 22 | typedef struct index_rec INDEX;
```

TEXT STATISTICS

```
   612 characters
    22 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     2 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    END_REC         14
    INDEX           22
    MAX_KEY         13      18
    MAX_LINE        12
    index_rec       16      22
    key             18
    pos             19
```

```
/* +++Date last modified: 05-Jul-1997 */

/**** init_globals(fp, names, types, ...)
**
** public domain by Raymond Gardner     Sept. 1991
**
** fp is a FILE * to the (already fopen'ed) file containing
**      initialization data
**  names is a space-separated list of names of globals (as they
**      are to appear in the data file)
**  types is a list of datatype characters, corresponding to names
**      i  for a pointer to integer
**      s  for a pointer to string (already allocated char array)
**      p  for a pointer to pointer to char (space will be malloc'd)
**    (NOTE: no whitespace allowed in types !!)
**  followed by var arg list of pointers to variables to init
*/

#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include "initvars.h"

int init_globals(FILE *fp, char *names, char *types, ...)
{
    char ln[LNSZ];
    char *p;
    va_list arglist;
    char *namep, *typep, name[40], *e;
    void *argp;
    int k;

    while ( fgets(ln, LNSZ, fp) ) {             /* read init file */
        while ( isspace(ln[0]) )            /* drop leading whitespace */
            memmove(ln, ln+1, strlen(ln));
        if ( ln[0] == 0 )                       /* skip if blank line */
            continue;
        p = strchr(ln, '=');                    /* find equal sign */
        if ( p == NULL )                        /* error if none */
            return -1;  /* or continue; */
        while ( p > ln && isspace(p[-1]) ) {    /* remove whitespace */
            memmove(p-1, p, strlen(p-1));       /*   before = sign */
            --p;
        }
        *p++ = 0;                               /* plug EOS over = sign */
        while ( isspace(p[0]) )             /* remove leading space on */
            memmove(p, p+1, strlen(p));         /*    init string */
        k = strlen(p) - 1;                      /* init string length */
        if ( k < 1 )
            return -1;

        if ( p[k] != '\n' )             /* if '\n' is missing, input */
            return -1;                  /*   exceeded buffer; error return */
        p[k] = 0;                       /* plug EOS over newline */

        va_start(arglist, types);       /* setup for arglist search */

        namep = names;                  /* init ptr to var names */
        typep = types;                  /* init ptr to var types */
        while ( *namep == ' ' )         /* skip blanks before namelist */
            ++namep;
        while ( *typep ) {          /* while any typelist items left...*/

            argp = (void *)va_arg(arglist, void *); /* get var arg */

            k = strcspn(namep, " ");        /* length of namelist entry */
            memmove(name, namep, k);        /* put into name hold area */
            name[k] = 0;                    /* terminate it */
            if ( strcmp(name, ln) != 0 ) { /* if it doesn't match... */
                namep += k;                 /* get next name */
                while ( *namep == ' ' )
                    ++namep;
                ++typep;                    /* get next type */
            } else {                        /* else name is found... */
                if ( *typep == 'i' ) {      /* if it's an int, init it */
                    *(int *)argp = atoi(p);
                } else if ( *typep == 's' || *typep == 'p' ) {
                    if ( *p == '"' ) {      /* is string in quotes? */
                        ++p;                /* skip leading quote, and */
                        e = strchr(p, '"'); /* look for trailing quote */
                        if ( e )            /* terminate string if found */
                            *e = 0;
                    }
                    if ( *typep == 'p' ) {      /* if it's a char *ptr */
                        e = malloc(strlen(p) + 1); /* get space */
                        if ( e == 0 ) {         /* error if no space */
                            return -1; /* call va_end(arglist); first? */
                        }
                        *(char **)argp = e;
                        strcpy(*(char **)argp, p);  /* copy in string */
                    } else                          /* must be char array */
                        strcpy(argp, p);            /* copy in string */
                } else {
                    return -1;                      /* bad type */
                }
                break;              /* break search; get next line */
            }
        }
        va_end(arglist);
```

```
    }
    return 0;
}

#ifdef TEST

#include "errors.h"                /* For cant()     */

int foo;
char bar[80];
int baz;
char *quux;

int main(int argc, char **argv)
{
    FILE *fp;
    int k;

    if ( argc < 2 ) {
        fprintf(stderr, "missing arg\n");
        exit(1);
    }

    fp = cant(argv[1], "r");
    k = init_globals(fp, "foo bar baz quux", "isip",
        &foo, bar, &baz, &quux);
    printf("k: %d\n", k);
    printf("foo: %d\nbar: <%s>\nbaz: %d\nquux: <%s>\n",
        foo, bar, baz, quux);
    fclose(fp);

    return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  5200 characters
   135 lines

LEXICAL STATISTICS

    46 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
    10 constants [character]
     9 constants [string]
    30 constants [numeric]

SYMBOL TABLE

    FILE            25     116
    LNSZ            27      34
    NULL            40
    TEST           105
    argc           114     119
    arglist         29      57      65     100
    argp            31      65      77      90      91      93
    argv           114     124
    atoi            77
    bar            110     126     129
    baz            111     126     129
    cant           124
    ctype           22
    e       30      81      82      83      86      87      90
    exit           121
    fclose         130
    fgets           34
    foo            109     126     129
    fp              25      34     116     124     125     130
    fprintf        120
    h       19      20      21      22
    init_globals            25     125
    isspace         35      42      47
    k       32      49      50      53      55      67      68      69      71     117     125     127
    ln              27      34      35     36+      37      39      42      70
    main           114
    malloc          86
    memmove         36      43      48      68
    name            30      68      69      70
    namep           30      59      61      62      67      68      71      72      73
    names           25      59
    p       28      39      40     42+     43+      44      46      47     48+      49      53      55
            77      79      80      81      86      91      93
    printf         127     128
    quux           112     126     129
    stdarg          20
    stderr         120
    stdlib          19
    strchr          39      81
    strcmp          70
    strcpy          91      93
    strcspn         67
    string          21
    strlen          36      43      48      49      86
    typep           30      60      63      74      76     78+      85
    types           25      57      60
    va_arg          65
    va_end         100
    va_list         29
    va_start        57
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header file for INITVARS.C
   5 | */
   6 |
   7 | #ifndef INITVARS__H
   8 | #define INITVARS__H
   9 |
  10 | #include <stdio.h>
  11 |
  12 | #define LNSZ 256
  13 |
  14 | int init_globals(FILE *fp, char *names, char *types, ...);
  15 |
  16 | #endif /* INITVARS__H */
```

TEXT STATISTICS

```
   269 characters
    16 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    FILE            14
    INITVARS__H              7       8
    LNSZ            12
    fp              14
    h       10
    init_globals            14
    names           14
    stdio           10
    types           14
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  Header for Int 2Eh interface
   5 | */
   6 |
   7 | #ifndef INT2E__H
   8 | #define INT2E__H
   9 |
  10 | #include "extkword.h"
  11 |
  12 | extern void CDECL _Int_2E(char *);
  13 | void C_Com_Call(char *string);
  14 |
  15 | #endif /* INT2E__H */
```

TEXT STATISTICS

```
   244 characters
    15 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CDECL           12
    C_Com_Call              13
    INT2E__H         7       8
    _Int_2E         12
    string          13
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPPETS functions for Intel CPUs and NDP's
  5 | */
  6 |
  7 | #ifndef INTEL__H
  8 | #define INTEL__H
  9 |
 10 | #include "sniptype.h"         /* For True_ and False_                   */
 11 | #include "extkword.h"         /* For CDECL                             */
 12 |
 13 | int CDECL cpu_ID(void);       /* Returns: AL - CPU family
 14 |                                               0 = 8086/8088
 15 |                                               1 = 80186
 16 |                                               2 = 80286
 17 |                                               3 = 80386
 18 |                                               4 = 80486
 19 |                                               5 = Pentium
 20 |                                               6 = Pentium Plus
 21 |                                         AH - Features bitmap
 22 |                                               01h = Has coprocessor
 23 |                                               02h = Is 386/287
 24 |                                               04h = Is 386SX
 25 |                                               08h = Is Cyrix
 26 |                                               10h = Is NEC
 27 |                                               20h = Is NexGen
 28 |                                               40h = is AMD
 29 |                                               80h = Is UMC            */
 30 |
 31 | extern CDECL save_8087(void);       /* Save coprocessor environment     */
 32 | extern CDECL restore_8087(void);    /* Restore coprocessor environment  */
 33 |
 34 | #endif /* INTEL__H */
```

TEXT STATISTICS

```
  1539 characters
    34 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CDECL           13      31      32
    INTEL__H         7       8
    cpu_ID          13
    restore_8087            32
    save_8087       31
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  IPOW.C - Raise a number to an integer power
 5 | **
 6 | **  public domain by Mark Stephen with suggestions by Keiichi Nakasato
 7 | */
 8 |
 9 | #include "snipmath.h"
10 |
11 | double ipow(double x, int n)        /* return x^n */
12 | {
13 |       double t = 1.0;
14 |
15 |       if (!n)
16 |             return 1.0;    /* At the top. 0**0 = 1 */
17 |       if (n < 0)
18 |       {
19 |             n = -n;
20 |             x = 1.0/x;  /* error if x == 0. Good                        */
21 |       }                 /* ZTC/SC returns inf, which is even better     */
22 |       if (x == 0.0)
23 |             return 0.0;
24 |       do
25 |       {
26 |             if (n & 1)
27 |                   t *= x;
28 |             n /= 2;     /* KN prefers if (n/=2) x*=x; This avoids an    */
29 |             x *= x;     /* unnecessary but benign multiplication on     */
30 |       } while (n);      /* the last pass, but the comparison is always  */
31 |       return t;         /* true _except_ on the last pass.              */
32 | }
33 |
34 | #ifdef TEST
35 |
36 | #include <stdio.h>
37 | #include <stdlib.h>
38 |
39 | #ifdef __WATCOMC__
40 |  #pragma off (unreferenced);
41 | #endif
42 | #ifdef __TURBOC__
43 |  #pragma argsused
44 | #endif
45 |
46 | main(int argc, char *argv[])
47 | {
48 |       double d;
49 |       int n;
50 |
51 |       d = atof(argv[1]);
52 |       n = atoi(argv[2]);
53 |       printf("%f^%d = %f\n", d, n, ipow(d, n) );
54 |       return 0;
55 | }
56 |
57 | #endif /* TEST */
```

TEXT STATISTICS

```
  1354 characters
    57 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     2 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

TEST		34								
__TURBOC__			42							
__WATCOMC__			39							
argc		46								
argsused		43								
argv		46	51	52						
atof		51								
atoi		52								
d	48	51	53+							
h	36	37								
ipow		11	53							
main		46								
n	11	15	17	19+	26	28	30	49	52	53+
off		40								
printf		53								
stdio		36								
stdlib		37								
t	13	27	31							
unreferenced			40							
x	11	20+	22	27	29+					

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  Public domain by Paul Schlyter, 27-Apr-1994
 5 | **
 6 | **  modified for SNIPPETS by Bob Stout
 7 | **
 8 | **  Pass: 0 for drive A:, 1 for drive B:, 2 for drive C:, etc.
 9 | **
10 | **  Returns: True_  if the drive is a CD-ROM
11 | **           False_ if the drive is not a CD-ROM
12 | **           Error_ if MSCDEX not installed
13 | */
14 |
15 | #include "dosfiles.h"
16 |
17 | Boolean_T isCDROMdrive(int drive)
18 | {
19 |       union REGS r;
20 |
21 |       r.x.ax = 0x1500;              /* First test for presence of MSCDEX */
22 |       r.x.bx = 0;
23 |       int86( 0x2F, &r, &r );
24 |       if ( r.x.bx == 0 )
25 |             return Error_;          /* MSCDEX not there                  */
26 |       r.x.ax = 0x150B;              /* MSCDEX driver check API           */
27 |       r.x.cx = drive - 'A';
28 |       int86( 0x2F, &r, &r );
29 |       return  r.x.ax != 0;          /* Drive is CDROM if AX nonzero      */
30 | }
```

TEXT STATISTICS

```
   891 characters
    30 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     1 constants [character]
     1 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       17
    Error_          25
    REGS            19
    ax              21      26      29
    bx              22      24
    cx              27
    drive           17      27
    int86           23      28
    isCDROMdrive            17
    r       19      21      22      23+     24      26      27      28+     29
    x       21      22      24      26      27      29
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  iscons()
  5 | **
  6 | **  A function to determine if a specified stream refers to the console.
  7 | **
  8 | **  Original Copyright 1988-1991 by Bob Stout as part of
  9 | **  the MicroFirm Function Library (MFL)
 10 | **
 11 | **  The user is granted a free limited license to use this source file
 12 | **  to create royalty-free programs, subject to the terms of the
 13 | **  license restrictions specified in the LICENSE.MFL file.
 14 | */
 15 |
 16 | #include <dos.h>
 17 | #include "dosfiles.h"
 18 |
 19 | int iscons(FILE *fp)
 20 | {
 21 |       union REGS regs;
 22 |
 23 |       regs.x.ax = 0x4400;
 24 |       regs.x.bx = (unsigned)fileno(fp);
 25 |       intdos(&regs, &regs);
 26 |       if (0 == (regs.x.ax & 0x80))
 27 |             return 0;
 28 |       return TOBOOL(regs.x.ax & 0x13);
 29 | }
 30 |
 31 | #ifdef TEST
 32 |
 33 | int main(void)
 34 | {
 35 |       fprintf(stderr, "stdin is%s redirected\n",
 36 |             iscons(stdin) ? " not" : "");
 37 |       fprintf(stderr, "stdout is%s redirected\n",
 38 |             iscons(stdout) ? " not" : "");
 39 |       return 0;
 40 | }
 41 |
 42 | #endif /* TEST */
```

TEXT STATISTICS

```
  1011 characters
    42 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     7 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    FILE            19
    REGS            21
    TEST            31
    TOBOOL          28
    ax              23      26      28
    bx              24
    dos             16
    fileno          24
    fp              19      24
    fprintf         35      37
    h       16
    intdos          25
    iscons          19      36      38
    main            33
    regs            21      23      24      25+     26      28
    stderr          35      37
    stdin           36
    stdout          38
    x       23      24      26      28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ISDST.C - Determine is a date falls within Daylight Savings Time.
**
**  The rule is that DST starts at 2:00 AM on the first Sunday of April
**  and ends at 2:00 AM on the last Sunday of October.
**
**  Original Copyright 1995 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include "sniptype.h"
#include "scaldate.h"

/*
**  Encode DST rules here - defaults to standard U.S. rules.
*/

unsigned    DST_start_mo = 4;            /* Month when DST starts      */
unsigned    DST_start_dt = 1;            /* Date when it can start     */
enum DOW_T  DST_start_dy = SUNDAY;       /* Day of week, or DOW_IGNORE */

unsigned    DST_stop_mo = 10;            /* Month when DST stops       */
unsigned    DST_stop_dt = 31;            /* Date when it can stop      */
enum DOW_T  DST_stop_dy = SUNDAY;        /* Day of week, or DOW_IGNORE */

/*
**  isDST()
**
**  Parameters: 1 - Year of interest.
**              2 - Month to check.
**              3 - Day to check.
**              4 - Pointer to storage for scalar date representation of
**                  the year's DST start date.
**              5 - Pointer to storage for scalar date representation of
**                  the year's DST stop date.
**
**  Returns: True_ if date is in DST range
**           False_ if date is not in DST  range
**           Error_ if date is invalid,
*/

Boolean_T isDST(unsigned  yr,
                unsigned  mo,
                unsigned  dy,
                long     *Start,
                long     *Stop)
{
      long date;

      if (!valiDate(yr, mo, dy))
            return Error_;
      else  date = ymd_to_scalar(yr, mo, dy);

      *Start = ymd_to_scalar(yr, DST_start_mo, DST_start_dt);
      *Stop  = ymd_to_scalar(yr, DST_stop_mo, DST_stop_dt);

      if (DST_start_dy != DOW_IGNORE)
      {
            while (DST_start_dy != (*Start % 7L))
                  ++*Start;
      }

      if (DST_stop_dy != DOW_IGNORE)
      {
            while (DST_stop_dy != (*Stop % 7L))
                  --*Stop;
      }

      return (date >= *Start && date < *Stop);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>
#include "datetime.h"

main(int argc, char *argv[])
{
      long Start, Stop;
      unsigned yr, mo, dy;
      Boolean_T retval;

      if (2 > argc)
      {
            puts("Usage: ISDST \"date_string\"");
            return EXIT_FAILURE;
      }
      if (Success_ != parse_date(argv[1], &yr, &mo, &dy, USA))
      {
            printf("Error parsing date \"%s\"\n", argv[1]);
            return EXIT_FAILURE;
      }

```

```
 101 |       if (Error_ == (retval = isDST(yr, mo, dy, &Start, &Stop)))
 102 |             printf("%d/%d,%d is invalid!\n", mo, dy, yr);
 103 |       else
 104 |       {
 105 |             printf("%d/%d/%d is%s within DST range\n\n", mo, dy, yr,
 106 |                   retval ? "" : " not");
 107 |             scalar_to_ymd(Start, &yr, &mo, &dy);
 108 |             printf("In %d, DST starts on %d/%d\n", yr, mo, dy);
 109 |             scalar_to_ymd(Stop, &yr, &mo, &dy);
 110 |             printf("In %d, DST stops on %d/%d\n", yr, mo, dy);
 111 |       }
 112 |
 113 |       return EXIT_SUCCESS;
 114 | }
 115 |
 116 | #endif /* TEST */
```

TEXT STATISTICS

```
  3453 characters
   116 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
    11 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       48      88
    DOW_IGNORE              63      69
    DOW_T           26      30
    DST_start_dt            25      60
    DST_start_dy            26      63      65
    DST_start_mo            24      60
    DST_stop_dt             29      61
    DST_stop_dy             30      69      71
    DST_stop_mo             28      61
    EXIT_FAILURE            93      98
    EXIT_SUCCESS           113
    Error_          57     101
    SUNDAY          26      30
    Start           51      60      65      66      75      86     101     107
    Stop            52      61      71      72      75      86     101     109
    Success_        95
    TEST            78
    USA             95
    argc            84      90
    argv            84      95      97
    date            54      58      75+
    dy              50      56      58      87      95     101     102     105     107     108     109
          110
    h        80     81
    isDST           48     101
    main            84
    mo              49      56      58      87      95     101     102     105     107     108     109
          110
    parse_date              95
    printf          97     102     105     108     110
    puts            92
    retval          88     101     106
    scalar_to_ymd          107     109
    stdio           80
    stdlib          81
    valiDate        56
    ymd_to_scalar           58      60      61
    yr              48      56      58      60      61      87      95     101     102     105     107
          108     109     110
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Find out if a FILE * is valid
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include "dosfiles.h"
 10 |
 11 | #ifdef __TURBOC__
 12 |  #define STREAM_BUF     _streams
 13 |  #define FCNT           FOPEN_MAX
 14 |  #define FLAG           flags
 15 | #elif defined(__WATCOMC__)
 16 |  #define STREAM_BUF     __iob
 17 |  #define FCNT           _NFILES
 18 |  #define FLAG           _flag
 19 | #else /* MSC, ZTC++ */
 20 |  #define STREAM_BUF     _iob
 21 |  #define FCNT           _NFILE
 22 |  #define FLAG           _flag
 23 | #endif
 24 |
 25 | int isfopen(FILE *fp)
 26 | {
 27 |       int i;
 28 |
 29 |       for (i = 0; i < FCNT; ++i)
 30 |       {
 31 |             if (0 != STREAM_BUF[i].FLAG && fp == &STREAM_BUF[i])
 32 |                   return True_;
 33 |       }
 34 |       return False_;
 35 | }
 36 |
 37 | #ifdef TEST
 38 |
 39 | main()
 40 | {
 41 |       printf("stdout is%s valid\n", isfopen(stdout) ? "":" not");
 42 |       printf("buffer #10 is%s valid\n", isfopen(&STREAM_BUF[9]) ? "":" not");
 43 |       return 0;
 44 | }
 45 |
 46 | #endif /*TEST */
```

TEXT STATISTICS

```
   956 characters
    46 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
     7 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    FCNT            13      17      21      29
    FILE            25
    FLAG            14      18      22      31
    FOPEN_MAX       13
    False_          34
    STREAM_BUF              12      16      20      31+     42
    TEST            37
    True_           32
    _NFILE          21
    _NFILES         17
    __TURBOC__              11
    __WATCOMC__             15
    __iob           16
    _flag           18      22
    _iob            20
    _streams        12
    defined         15
    flags           14
    fp              25      31
    i       27      29+     31+
    isfopen         25      41      42
    main            39
    printf          41      42
    stdout          41
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  ISISBN.C - Validate International Standard Book Numbers (ISBNs)
  5 | **
  6 | **  public domain by Maynard Hogg
  7 | */
  8 |
  9 | #include <ctype.h>
 10 | #include "isisbn.h"
 11 |
 12 | int isbn2(char *str)
 13 | {
 14 |       int i = 0;
 15 |       int test = 0;
 16 |       int c;
 17 |
 18 |       while ('\0' != (c = *str++))
 19 |       {
 20 |             if (isdigit(c))
 21 |                   c -= '0';
 22 |             else if (i == 9 && 'X' == c)
 23 |                   c = 10;
 24 |             else continue;
 25 |             test += c * ++i;
 26 |       }
 27 |       return (i == 10 && test % 11 == 0);
 28 | }
 29 |
 30 | #ifdef TEST
 31 |
 32 | #include <stdio.h>
 33 |
 34 | main(int argc, char *argv[])
 35 | {
 36 |       while (--argc)
 37 |       {
 38 |             int r = isbn2(*++argv);
 39 |
 40 |             printf("%s is%s a valid ISBN number\n", *argv, r ? "" : " not");
 41 |       }
 42 |       return 0;
 43 | }
 44 |
 45 | #endif /* TEST */
```

TEXT STATISTICS

```
   840 characters
    45 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     3 constants [character]
     4 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    TEST            30
    argc            34      36
    argv            34      38      40
    c       16      18      20      21      22      23      25
    ctype            9
    h        9      32
    i       14      22      25      27
    isbn2           12      38
    isdigit         20
    main            34
    printf          40
    r       38      40
    stdio           32
    str             12      18
    test            15      25      27
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header file for ISISBN.C
   5 | */
   6 |
   7 | #ifndef ISISBN__H
   8 | #define ISISBN__H
   9 |
  10 | int isbn2(char *str);
  11 |
  12 | #endif /* ISISBN__H */
```

TEXT STATISTICS

```
   182 characters
    12 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    ISISBN__H        7       8
    isbn2           10
    str             10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*--------------------------------------------------------------------------*/
  4 | /*  determine_drive_type -- Public Domain code from Bob Dolan               */
  5 | /*                                                                          */
  6 | /*  INPUT:  the drive number ( 0=current, 1=A:, 2=B:, etc. )                */
  7 | /*  OUTPUT: drive type ( 0=physical drive, 1=Network drive, 2=RamDisk ) */
  8 | /*--------------------------------------------------------------------------*/
  9 |
 10 | #include <dos.h>
 11 | #include "dosfiles.h"
 12 |
 13 | int drive_type(int dr)
 14 | {
 15 |       union REGS regs;
 16 |
 17 |       regs.x.ax = 0x4409;     /* IOCTL func 9 */
 18 |       regs.h.bl = (unsigned char)dr;
 19 |       int86(0x21, &regs, &regs);
 20 |       if (!regs.x.cflag)
 21 |       {
 22 |             if (regs.x.dx & 0x1000)
 23 |                   return 1;   /* Network drive */
 24 |
 25 |             else if (regs.x.dx == 0x0800)
 26 |                   return 2;   /* RAMdisk */
 27 |       }
 28 |
 29 |       return 0;   /* physical drive */
 30 | }
 31 |
 32 | #ifdef TEST
 33 |
 34 | #include <stdio.h>
 35 | #include <stdlib.h>
 36 | #include <ctype.h>
 37 |
 38 | int main(int argc, char *argv[])
 39 | {
 40 |       int dr = 0;
 41 |
 42 |       if (1 < argc)
 43 |             dr = toupper(*argv[1]) - '@';
 44 |       printf ("drive_type(%d) = %d\n", dr, drive_type(dr));
 45 |       return 0;
 46 | }
 47 |
 48 | #endif /* TEST */
```

TEXT STATISTICS

```
  1300 characters
    48 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     1 constants [character]
     2 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    REGS            15
    TEST            32
    argc            38      42
    argv            38      43
    ax              17
    bl              18
    cflag           20
    ctype           36
    dos             10
    dr              13      18      40      43      44+
    drive_type              13      44
    dx              22      25
    h       10      18      34      35      36
    int86           19
    main            38
    printf          44
    regs            15      17      18      19+     20      22      25
    stdio           34
    stdlib          35
    toupper         43
    x       17      20      22      25
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #include <stdio.h>
  4 | #include "snipmath.h"
  5 |
  6 | int ispow2(int x)
  7 | {
  8 |       return! ((~(~0U>>1)|x)&x -1) ;
  9 | }
 10 |
 11 | #ifdef TEST
 12 |
 13 | int main(void)
 14 | {
 15 |       int i;
 16 |
 17 |       for (i = 0; i < 256; ++i)
 18 |             printf("%3d: %d\n", i, ispow2(i));
 19 |       return 0;
 20 | }
 21 |
 22 | #endif /* TEST */
```

TEXT STATISTICS

```
   326 characters
    22 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     2 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    TEST            11
    h        3
    i       15      17+     18+
    ispow2           6      18
    main            13
    printf          18
    stdio            3
    x        6       8+
```

```
/* +++Date last modified: 05-Jul-1997 */

#include <string.h>
#include "snipmath.h"

#define BITSPERLONG 32

#define TOP2BITS(x) ((x & (3L << (BITSPERLONG-2))) >> (BITSPERLONG-2))


/* usqrt:
    ENTRY x: unsigned long
    EXIT  returns floor(sqrt(x) * pow(2, BITSPERLONG/2))

    Since the square root never uses more than half the bits
    of the input, we use the other half of the bits to contain
    extra bits of precision after the binary point.

    EXAMPLE
        suppose BITSPERLONG = 32
        then    usqrt(144) = 786432 = 12 * 65536
                usqrt(32) = 370727 = 5.66 * 65536

    NOTES
        (1) change BITSPERLONG to BITSPERLONG/2 if you do not want
            the answer scaled.  Indeed, if you want n bits of
            precision after the binary point, use BITSPERLONG/2+n.
            The code assumes that BITSPERLONG is even.
        (2) This is really better off being written in assembly.
            The line marked below is really a "arithmetic shift left"
            on the double-long value with r in the upper half
            and x in the lower half.  This operation is typically
            expressible in only one or two assembly instructions.
        (3) Unrolling this loop is probably not a bad idea.

    ALGORITHM
        The calculations are the base-two analogue of the square
        root algorithm we all learned in grammar school.  Since we're
        in base 2, there is only one nontrivial trial multiplier.

        Notice that absolutely no multiplications or divisions are performed.
        This means it'll be fast on a wide range of processors.
*/

void usqrt(unsigned long x, struct int_sqrt *q)
{
      unsigned long a = 0L;                   /* accumulator      */
      unsigned long r = 0L;                   /* remainder        */
      unsigned long e = 0L;                   /* trial product    */

      int i;

      for (i = 0; i < BITSPERLONG; i++)   /* NOTE 1 */
      {
            r = (r << 2) + TOP2BITS(x); x <<= 2; /* NOTE 2 */
            a <<= 1;
            e = (a << 1) + 1;
            if (r >= e)
            {
                  r -= e;
                  a++;
            }
      }
      memcpy(q, &a, sizeof(long));
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(void)
{
      int i;
      unsigned long l = 0x3fed0169L;
      struct int_sqrt q;

      for (i = 0; i < 101; ++i)
      {
            usqrt(i, &q);
            printf("sqrt(%3d) = %2d, remainder = %2d\n",
                  i, q.sqrt, q.frac);
      }
      usqrt(l, &q);
      printf("\nsqrt(%lX) = %X, remainder = %X\n", l, q.sqrt, q.frac);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2735 characters
    89 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     3 constants [string]
    17 constants [numeric]
```

SYMBOL TABLE

```
    BITSPERLONG              6       8+     53
    TEST            67
    TOP2BITS         8      55
    a       47      56      57      61      64
    e       49      57      58      60
    frac            82      85
    h        3      69      70
    i       51      53+     74      78+     80      82
    int_sqrt        45      76
    l       75      84      85
    main            72
    memcpy          64
    printf          81      85
    q       45      64      76      80      82+     84      85+
    r       48      55+     58      60
    sqrt            82      85
    stdio           69
    stdlib          70
    string           3
    usqrt           45      80      84
    x        8+     45      55+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  isRamDsk() - Determine if a drive is a RAM disk
**
**  Call with drive letter ('a' - 'z', 'A' - 'Z')
**
**  Returns True_, False_, or Error_
**
**  Uses ABSDISKC.C, ABSDISK.ASM, and DOS5BOOT.H from SNIPPETS
**  (Note: The relevant parts of the structure in DOS5BOOT.H are
**   also applicable to lower version numbers of DOS)
**
**  Public domain by Bob Stout
*/

#include <stdlib.h>
#include <ctype.h>
#include <dos.h>
#include "dos5boot.h"
#include "dosfiles.h"
#include "snpdskio.h"

Boolean_T isRamDsk(unsigned char drive)
{
      union REGS regs;
      B_REC buffer;

      regs.x.ax = 0x4408;           /* Not if removable     */
      regs.h.bl = (unsigned)toupper(drive) - (unsigned char)'@';
      intdos(&regs, &regs);
      if (0 == regs.x.ax)
            return False_;
      if (AbsDiskRead(toupper(drive) - 'A', 1, 0, &buffer))
            return Error_;
      return (1 == buffer.bsFATs);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      if (2 > argc)
      {
            puts("Syntax: ISRAMDSK drive_letter");
            return EXIT_FAILURE;
      }
      printf("Drive %c: is%s a RAM drive\n", toupper(*argv[1]),
            isRamDsk(*argv[1]) ? "" : " not");
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1380 characters
    56 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     2 constants [character]
     7 constants [string]
     8 constants [numeric]

SYMBOL TABLE

    AbsDiskRead             34
    B_REC           27
    Boolean_T       24
    EXIT_FAILURE            49
    EXIT_SUCCESS            53
    Error_          35
    False_          33
    REGS            26
    TEST            39
    argc            44      46
    argv            44      51      52
    ax              29      32
    bl              30
    bsFATs          36
    buffer          27      34      36
    ctype           18
    dos             19
    drive           24      30      34
    h       17      18      19      30      41      42
    intdos          31
    isRamDsk        24      52
    main            44
    printf          51
    puts            48
    regs            26      29      30      31+     32
    stdio           41
    stdlib          17      42
    toupper         30      34      51
    x       29      32
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** is_share() - This is a routine that allows a program to determine
**              if file-sharing is enabled at run-time.
**
**              What does this code do?  First - it checks to make sure
**              it is running under DOS 3.0+ - otherwise - no sharing.
**              Next, it opens the program itself (the .EXE file) by using
**              "argv[0]".  This argument points to the actual program name
**              complete with the path under DOS 3.0 or later.  It then
**              attempts to lock the first 500 bytes of the program on
**              disk.  If successful (i.e. return != -1), it unlocks the
**              locked bytes and closes the file (actually the unlock is
**              superfluous since closing the file releases all locks) and
**              returns the a "TRUE" (1) result.  If it fails, it closes
**              the .EXE file and returns a "FALSE" (0) result.  Note that
**              this does not depend on opening a file in shared mode to
**              test it.
**
** Revision History:
**
** 08/03/93  Original:  "is_sharing()" by Mike Ratledge of fidonet
** 10/20/93  Revision:  revised for library
** 04/03/94  Revision:  "Portabilized" for SNIPPETS by Bob Stout
*/

#include <stdio.h>
#include <io.h>
#include <dos.h>
#include "snpdosys.h"
#include "errors.h"

#if !defined(__TURBOC__)
 #include <stdlib.h>
 #include <sys\locking.h>

 int lock(int fp, long ofs, long lng)
 {
       lseek(fp,ofs,SEEK_SET);
       return locking(fp,LK_LOCK,lng);
 }

 int unlock(int fp, long ofs, long lng)
 {
       lseek(fp,ofs,SEEK_SET);
       return locking(fp,LK_UNLCK,lng);
 }
#endif

int is_share(char *arg)
{
      FILE *exe;

      if (_osmajor < 3)
            return(0);

      exe = cant(arg, "rb");

      if (0 == lock(fileno(exe), 0L, 500L))
      {
            unlock(fileno(exe), 0L, 500L);
            fclose(exe);
            return(1);
      }

      fclose(exe);
      return(0);
}

#ifdef TEST

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif

#ifdef __TURBOC__
 #pragma argsused
#endif

main(int argc, char *argv[])
{
      int sharing = is_share(argv[0]);

      printf("File sharing is%s enabled\n", sharing ? "" : " not");
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2383 characters
    89 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    17 preprocessor instructions
     0 constants [character]
     6 constants [string]
    11 constants [numeric]

SYMBOL TABLE

    FILE            53
    LK_LOCK         41
    LK_UNLCK        47
    SEEK_SET        40      46
    TEST            71
    __TURBOC__              34      77
    __WATCOMC__             73
    _osmajor        55
    arg             51      58
    argc            81
    argsused        78
    argv            81      83
    cant            58
    defined         34
    dos             30
    exe             53      58      60      62      63      67
    fclose          63      67
    fileno          60      62
    fp              38      40      41      44      46      47
    h       28      29      30      35      36
    io              29
    is_share        51      83
    lng             38      41      44      47
    lock            38      60
    locking         41      47
    lseek           40      46
    main            81
    off             74
    ofs             38      40      44      46
    printf          85
    sharing         83      85
    stdio           28
    stdlib          35
    sys\locking             36
    unlock          44      62
    unreferenced            74
```

```
/* +++Date last modified: 05-Jul-1997 */

/*-------------------------[ IsShift ]--------------------------*/
/*             Determine whether a shift key is depressed       */
/*                public domain snippet by Jeff Dunlop          */
/*  Revisions:                                                  */
/*  30-Mar-96  Ed Blackman  OS/2 mods, added #defines for keys, */
/*                          wrapped platform dependant actions  */
/*                          in a macro, added some discussion   */
/*-------------------------------------------------------------*/
/* local:                                                      */
/*      kf =                                                   */
/*          DOS: pointer to bios shift key area                */
/*          OS2: pointer to previously stored key info struct  */
/* return:                                                     */
/*      non-zero if shift key is depressed                     */
/*-------------------------------------------------------------*/

#include "extkword.h"
#include "snipkbio.h"
#include "mk_fp.h"

#ifdef __OS2__
KBDKEYINFO ki;                          /* holds key status */
#endif /* !__OS2__ */

int IsLeftShift(void)
{
      unsigned short FAR* kf = peekkey();

      return (*kf & LEFT_SHIFT);
}

int IsRightShift(void)
{
      unsigned short FAR* kf = peekkey();

      return (*kf & RIGHT_SHIFT);
}

int IsShift(void)
{
      return (IsLeftShift()) || (IsRightShift());
}

int IsLeftAlt(void)
{
      unsigned short FAR* kf = peekkey();

      return (*kf & EITHER_ALT) && (*kf & LEFT_ALT);
}

int IsRightAlt(void)
{
      unsigned short FAR* kf = peekkey();

/* 30-Mar-96 - EBB: In DOS there is no direct way to tell if the right
** Alt key is depressed. DOS sets a status bit if either Alt key was
** pressed, and another if the left Alt key was pressed.  By
** elimination, we can say that the right Alt key was pressed if one of
** the Alt keys was pressed and it wasn't the left one.
*/
    return (*kf & EITHER_ALT) && !(*kf & LEFT_ALT);
}

int IsAlt(void)
{
      return (IsLeftAlt()) || (IsRightAlt());
}

int IsLeftCtl(void)
{
      unsigned short FAR* kf = peekkey();

      return (*kf & EITHER_CTL) && (*kf & LEFT_CTL);
}

int IsRightCtl(void)
{
    unsigned short FAR* kf = peekkey();

/* 30-Mar-96 - EBB: The discussion about the right Alt key in
** IsRightAlt() also applies to the right Ctl key
*/

      return (*kf & EITHER_CTL) && !(*kf & LEFT_CTL);
}

int IsCtl(void)
{
      return (IsLeftCtl()) || (IsRightCtl());
}

int IsSysRq(void)
{
      unsigned short FAR* kf = peekkey();

      return (*kf & SYSRQ);
}
```

TEXT STATISTICS

2754 characters
99 lines

LEXICAL STATISTICS

20 comments [std-C]
0 comments [C++]
5 preprocessor instructions
0 constants [character]
3 constants [string]
0 constants [numeric]

SYMBOL TABLE

```
EITHER_ALT              50      63
EITHER_CTL              75      86
FAR             29      36      48      55      73      80      96
IsAlt           66
IsCtl           89
IsLeftAlt       46      68
IsLeftCtl       71      91
IsLeftShift             27      43
IsRightAlt              53      68
IsRightCtl              78      91
IsRightShift            34      43
IsShift         41
IsSysRq         94
KBDKEYINFO              24
LEFT_ALT        50      63
LEFT_CTL        75      86
LEFT_SHIFT              31
RIGHT_SHIFT             38
SYSRQ           98
__OS2__         23
kf              29      31      36      38      48      50+     55      63+     73      75+     80
        86+     96      98
ki              24
peekkey         29      36      48      55      73      80      96
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ISWPROT.C - Detect if floppy drive is write protected
**
**  public domain by Bob Stout w/ corrections & additions by Wayne King
*/

#include <dos.h>
#include "dosfiles.h"

/*
**  isWprot()
**
**  Parameters: 1 - Drive number (A: = 0, B: = 1)
**
**  Returns: -1 - Error
**            0 - Not write protected
**            1 write protected
**
**  Note: If drive door is open, an error is returned but the critical
**        error handler is NOT tripped
*/

int isWprot(int drive)
{
      union REGS regs;
      struct SREGS sregs;
      char buf[512], FAR *bufptr = (char FAR *)buf;   /* Needed by MSC  */

      /* First read sector 0  */

      segread(&sregs);
      regs.x.ax = 0x201;
      regs.x.cx = 1;
      regs.x.dx = drive & 0x7f;
      sregs.es  = FP_SEG(bufptr);
      regs.x.bx = FP_OFF(bufptr);
      int86x(0x13, &regs, &regs, &sregs);
      if (regs.x.cflag && regs.h.ah != 6)
      {
            regs.h.ah = 0x00;         /* reset diskette subsystem */
            regs.h.dl = drive & 0x7f;
            int86x(0x13, &regs, &regs, &sregs);
            return -1;
      }

      /* Try to write it back */

      segread(&sregs);
      regs.x.ax = 0x301;
      regs.x.cx = 1;
      regs.x.dx = drive & 0x7f;
      sregs.es  = FP_SEG(bufptr);
      regs.x.bx = FP_OFF(bufptr);
      int86x(0x13, &regs, &regs, &sregs);
      return (3 == regs.h.ah);
}

#ifdef TEST

#include <stdio.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
      int drive;

      if (2 > argc)
      {
            puts("Usage: ISWPROT drive_letter");
            return -1;
      }
      drive = toupper(argv[1][0]) - 'A';
      printf("isWprot(%c:) returned %d\n", drive + 'A', isWprot(drive));
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1860 characters
    79 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     2 constants [character]
     3 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    FAR             29+
    FP_OFF          38      55
    FP_SEG          37      54
    REGS            27
    SREGS           28
    TEST            60
    ah              40      42      57
    argc            65      69
    argv            65      74
    ax              34      51
    buf             29+
    bufptr          29      37      38      54      55
    bx              38      55
    cflag           40
    ctype           63
    cx              35      52
    dl              43
    dos              9
    drive           25      36      43      53      67      74      75+
    dx              36      53
    es              37      54
    h         9     40      42      43      57      62      63
    int86x          39      44      56
    isWprot         25      75
    main            65
    printf          75
    puts            71
    regs            27      34      35      36      38      39+     40+     42      43      44+     51
           52       53      55      56+     57
    segread         33      50
    sregs           28      33      37      39      44      50      54      56
    stdio           62
    toupper         74
    x       34      35      36      38      40      51      52      53      55
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  ISXKBRD.C - public domain by Ed Kowalski.
**
**  isxkeybrd() - detects enhanced kbd
*/

#include <dos.h>
#include "snipkbio.h"

/*
**  Check for enhanced keyboard support.
*/

int isxkeybrd(void)
{
      union REGS rg;
      unsigned kbdflags;

      rg.h.ah = 0x02;               /* check BIOS supports enhanced kbd */
      int86(0x16, &rg, &rg);        /* get kbd flags                    */
      kbdflags = rg.h.al;

      /* mess 'em up, get enhanced flags */

      rg.x.ax = 0x1200 + kbdflags ^ 0xff;
      int86(0x16, &rg, &rg);
      if (rg.h.al == kbdflags)      /* BIOS supports enhanced keyboard  */
      {
            /* if bit 4 at 40:96h is set machine has an enhanced kbd */

            if ((*(( char far *) 0x400096L) & 0x10))
                  return 1;         /* enhanced keyboard present        */
      }
      return 0;                     /* don't use enhanced keyboard calls */
}

#ifdef TEST

#include <stdio.h>
main()
{
      if (isxkeybrd())
            puts( "Enhanced Keyboard supported" );
      else  puts( "Enhanced Keyboard NOT supported ");
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1215 characters
    50 lines

LEXICAL STATISTICS

    11 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     3 constants [string]
    10 constants [numeric]

SYMBOL TABLE

    REGS            18
    TEST            39
    ah              21
    al              23      29
    ax              27
    dos              9
    far             33
    h        9      21      23      29      41
    int86           22      28
    isxkeybrd       16      44
    kbdflags        19      23      27      29
    main            42
    puts            45      46
    rg              18      21      22+     23      27      28+     29
    stdio           41
    x       27
```

```
/* +++Date last modified: 05-Jul-1997 */

/* jdn_l.c -- Julian Day Number computation
**
** public domain Julian Day Number functions
**
** Based on formulae originally posted by
**    Tom Van Flandern / Washington, DC / metares@well.sf.ca.us
**       in the UseNet newsgroup sci.astro.
**    Reposted 14 May 1991 in FidoNet C Echo conference by
**       Paul Schlyter (Stockholm)
** Minor corrections, added JDN to julian, and recast into C by
**    Raymond Gardner  Englewood, Colorado
**
** Synopsis:
**      long ymd_to_jdnl(int year, int month, int day, int julian_flag)
**      void jdnl_to_ymd(long jdnl, int *year, int *month, int *day,
**                                                      int julian_flag)
**      year is negative if BC
**      if julian_flag is >  0, use Julian calendar
**      if julian_flag is == 0, use Gregorian calendar
**      if julian_flag is <  0, routines decide based on date
**
** These routines convert Gregorian and Julian calendar dates to and
** from Julian Day Numbers.  Julian Day Numbers (JDN) are used by
** astronomers as a date/time measure independent of calendars and
** convenient for computing the elapsed time between dates.  The JDN
** for any date/time is the number of days (including fractional
** days) elapsed since noon, 1 Jan 4713 BC.  Julian Day Numbers were
** originated by Joseph Scaliger in 1582 and named after his father
** Julius, not after Julius Caesar.  They are not related to the
** Julian calendar.
**
** For dates from 1 Jan 4713 BC thru 12 Dec Feb 32766 AD, ymd_to_jdnl()
** will give the JDN for noon on that date.  jdnl_to_ymd() will compute
** the year, month, and day from the JDN.  Years BC are given (and
** returned) as negative numbers.  Note that there is no year 0 BC;
** the day before 1 Jan 1 AD is 31 Dec 1 BC.  Note also that 1 BC,
** 5 BC, etc. are leap years.
**
** Pope Gregory XIII decreed that the Julian calendar would end on
** 4 Oct 1582 AD and that the next day would be 15 Oct 1582 in the
** Gregorian Calendar.  The only other change is that centesimal
** years (years ending in 00) would no longer be leap years
** unless divisible by 400.  Britain and its possessions and
** colonies continued to use the Julian calendar up until 2 Sep
** 1752, when the next day became 14 Sep 1752 in the Gregorian
** Calendar.  These routines can be compiled to use either
** convention.  By default, the British convention will be used.
** Simply #define PAPAL to use Pope Gregory's convention.
**
** Each routine takes, as its last argument, a flag to indicate
** whether to use the Julian or Gregorian calendar convention.  If
** this flag is negative, the routines decide based on the date
** itself, using the changeover date described in the preceding
** paragraph.  If the flag is zero, Gregorian conventions will be used,
** and if the flag is positive, Julian conventions will be used.
**
** Proper JDN's are always floating point values so as to include the
** time as well as the date. These functions avoid the overhead of
** floating point math by computing only the integral value of the JDN.
*/

#include "datetime.h"

#ifdef PAPAL                    /* Pope Gregory XIII's decree */
#define LASTJULDATE 15821004L   /* last day to use Julian calendar */
#define LASTJULJDN  2299160L    /* jdn of same */
#else                           /* British-American usage */
#define LASTJULDATE 17520902L   /* last day to use Julian calendar */
#define LASTJULJDN  2361221L    /* jdn of same */
#endif


long ymd_to_jdnl(int y, int m, int d, int julian)
{
        long jdn;

        if (julian < 0)         /* set Julian flag if auto set */
                julian = (((y * 100L) + m) * 100 + d  <=  LASTJULDATE);

        if (y < 0)              /* adjust BC year */
                y++;

        if (julian)
                jdn = 367L * y - 7 * (y + 5001L + (m - 9) / 7) / 4
                + 275 * m / 9 + d + 1729777L;
        else
                jdn = (long)(d - 32076)
                + 1461L * (y + 4800L + (m - 14) / 12) / 4
                + 367 * (m - 2 - (m - 14) / 12 * 12) / 12
                - 3 * ((y + 4900L + (m - 14) / 12) / 100) / 4
                + 1;            /* correction by rdg */

        return jdn;
}


void jdnl_to_ymd(long jdn, int *yy, int *mm, int *dd, int julian)
{
```

```
101 |         long x, z, m, d, y;
102 |         long daysPer400Years = 146097L;
103 |         long fudgedDaysPer4000Years = 1460970L + 31;
104 |
105 |         if (julian < 0)                 /* set Julian flag if auto set */
106 |                 julian = (jdn <= LASTJULJDN);
107 |
108 |         x = jdn + 68569L;
109 |         if ( julian )
110 |         {
111 |                 x += 38;
112 |                 daysPer400Years = 146100L;
113 |                 fudgedDaysPer4000Years = 1461000L + 1;
114 |         }
115 |         z = 4 * x / daysPer400Years;
116 |         x = x - (daysPer400Years * z + 3) / 4;
117 |         y = 4000 * (x + 1) / fudgedDaysPer4000Years;
118 |         x = x - 1461 * y / 4 + 31;
119 |         m = 80 * x / 2447;
120 |         d = x - 2447 * m / 80;
121 |         x = m / 11;
122 |         m = m + 2 - 12 * x;
123 |         y = 100 * (z - 49) + y + x;
124 |
125 |         *yy = (int)y;
126 |         *mm = (int)m;
127 |         *dd = (int)d;
128 |
129 |         if (*yy <= 0)                   /* adjust BC years */
130 |                 (*yy)--;
131 | }
132 |
133 | #ifdef TEST
134 |
135 | #include <stdio.h>
136 | #include <stdlib.h>
137 |
138 | int main(int argc, char *argv[])
139 | {
140 |       int day, yr, mo;
141 |       char *months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
142 |                         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
143 |       long jdn;
144 |
145 |       if (3 > argc)
146 |       {
147 |             puts("Usage: JDN d m y");
148 |             puts("where: d = day (1 - 31)");
149 |             puts("       m = month (1 - 12)");
150 |             puts("       y = year (1 - 99, 1800 - 3000)");
151 |             return -1;
152 |       }
153 |
154 |       yr  = atoi(argv[3]);
155 |       mo  = atoi(argv[2]);
156 |       day = atoi(argv[1]);
157 |
158 |       if (mo < 1 || 12 < mo)
159 |             return -1;
160 |
161 |       if (day < 1 || 31 < day)
162 |             return -1;
163 |
164 |       if (100 > yr)
165 |             yr += 1900;
166 |
167 |       printf("User specified %d-%s-%d...\n\n", day, months[mo - 1], yr);
168 |
169 |       jdn = ymd_to_jdnl(yr, mo, day, 0);
170 |       jdnl_to_ymd(jdn, &yr, &mo, &day, 0);
171 |
172 |       printf("jdn_l(%d-%s-%d) = %ld\n", day, months[mo - 1], yr, jdn);
173 |       return 0;
174 | }
175 |
176 | #endif /* TEST */
```

TEXT STATISTICS

```
  6395 characters
   176 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
    19 constants [string]
    81 constants [numeric]
```

SYMBOL TABLE

```
    LASTJULDATE             67      70      80
    LASTJULJDN              68      71     106
    PAPAL           66
    TEST           133
    argc           138     145
    argv           138     154     155     156
    atoi           154     155     156
    d       75      80      87      89     101     120     127
    day            140     156     161+    167     169     170     172
    daysPer400Years         102     112     115     116
    dd              99     127
    fudgedDaysPer4000Years          103     113     117
    h      135     136
    jdn             77      86      89      95      99     106     108     143     169     170     172
    jdnl_to_ymd             99     170
    julian          75      79      80      85      99     105     106     109
    m       75      80      86      87      90      91+     92     101     119     120     121     122+
           126
    main           138
    mm              99     126
    mo             140     155     158+    167     169     170     172
    months         141     167     172
    printf         167     172
    puts           147     148     149     150
    stdio          135
    stdlib         136
    x      101     108     111     115     116+    117     118+    119     120     121     122     123
    y       75      80      82      83      86+     90      92     101     117     118     123+    125
    ymd_to_jdnl             75     169
    yr             140     154     164     165     167     169     170     172
    yy              99     125     129     130
    z      101     115     116     123
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  JGREP.C - A utility to search files for text.
**
**  public domain by Jerry Coffin
**
**  Link with wildargs.obj (Borland), setargv.obj (Microsoft), _mainX.obj
**  (Symantec/Zortech), or wildargX.obj (Watcom) which allows you to pass
**  wildcards on the command line.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>
#include "extkword.h"

#define LINELEN 1024
#define BUFSIZE 32767

#if defined(_QC) || defined(_MSC_VER)
    void CDECL _setenvp(void) {}
    void CDECL _nullcheck(void) {}
#endif

enum { FALSE, TRUE };

typedef unsigned char uchar;

static size_t table[UCHAR_MAX+1];
static size_t len;
static char *string=NULL;

void init_find(char *new_string)
{
      size_t i;

      if (NULL != string)
            free(string);
      string = strdup(new_string);
      len = strlen(string);

      for (i=0;i<=UCHAR_MAX;i++)
            table[i]=len;
      for (i=0;i<len;i++)
            table[string[i]]=len-i-1;
}

char *find_case(char *string2)
{
      size_t limit = strlen(string2);
      size_t shift;
      size_t pos=len-1;
      char *here;

      while (pos < limit)
      {
            while( pos < limit && (shift=table[(uchar)string2[pos]])>0)
                  pos+=shift;
            if (0==shift)
            {
                  if (!memcmp(string,here=string2+pos-len+1,len))
                        return(here);
                  else  pos++;
            }
      }
      return NULL;
}

char *find_no_case(char *string2)
{
      size_t limit = strlen(string2);
      size_t shift;
      size_t pos=len-1;
      char *here;

      while (pos < limit)
      {
            while( pos < limit &&
                  (shift=table[(uchar)toupper(string2[pos])])>0)
            {
                  pos+=shift;
            }
            if (0==shift)
            {
                  if (!memicmp(string,here=string2+pos-len+1,len))
                        return(here);
                  else  pos++;
            }
      }
      return NULL;
}

char *( *find)    (char *)=find_case;

int main(int argc, char **argv)
{
      int         k, first;
```

```
101 |       unsigned    line;
102 |       int line_numbers=FALSE, reverse=FALSE, print_file_name=FALSE;
103 |       int no_case = TRUE;
104 |       char line_buffer[LINELEN],*string,c;
105 |       FILE *infile=NULL;
106 |       static char buffer[BUFSIZE];
107 |
108 |       while (argc > 1 && (c=argv[1][0])=='/' || c=='-')
109 |       {
110 |             switch(tolower(argv[1][1]))
111 |             {
112 |             case 'c':
113 |             case 'y':
114 |                   no_case=FALSE;
115 |                   break;
116 |
117 |             case 'f':
118 |                   print_file_name=TRUE;
119 |
120 |             case 'n':
121 |                   line_numbers=TRUE;
122 |                   break;
123 |
124 |             case 'v':
125 |                   reverse = TRUE;
126 |                   break;
127 |
128 |             default:
129 |                   fprintf(stderr,"unknown switch -%c",argv[1][1]);
130 |             }
131 |             argv++;
132 |             argc--;
133 |       }
134 |       if (argc < 3)
135 |       {
136 |             fprintf(stderr ,"\nsyntax: find [-c|y][-n][-v][-f] "
137 |                   "string filename ..."
138 |                   "\n\t-c | -y : make case significant ('c' != 'C')"
139 |                   "\n\t-n      : number lines"
140 |                   "\n\t-f      : place file name before lines ( forces -n)"
141 |                   "\n\t-v      : print lines that don't match"
142 |                   "\n - by itself to read from standard input");
143 |             return(1);
144 |       }
145 |
146 |       string=argv[1];
147 |       if (no_case)
148 |       {
149 |             strupr(string);
150 |             find=find_no_case;
151 |       }
152 |       init_find(string);
153 |       for (k=2;k<argc;k++)
154 |       {
155 |             if (infile != NULL)
156 |                   fclose(infile);
157 |             if ((infile=fopen(argv[k],"r"))==NULL)
158 |                   continue;
159 |             first=TRUE;
160 |             setvbuf(infile,buffer,_IOFBF,BUFSIZE);
161 |             line=0;
162 |             while (fgets(line_buffer,LINELEN,infile)!=NULL)
163 |             {
164 |                   line ++;
165 |                   if ((NULL!=find(line_buffer)) ^ reverse)
166 |                   {
167 |                         if (first && !print_file_name )
168 |                         {
169 |                               printf("\n\t%s:\n",argv[k]);
170 |                               first = FALSE;
171 |                         }
172 |                         if (print_file_name)
173 |                               printf(" %s ",argv[k]);
174 |                         if (line_numbers)
175 |                               printf("%d:",line);
176 |                         printf("%s",line_buffer);
177 |                   }
178 |             }
179 |       }
180 |       return 0;
181 | }
```

```
TEXT STATISTICS

  4766 characters
   181 lines

LEXICAL STATISTICS

     2 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     7 constants [character]
    14 constants [string]
    27 constants [numeric]

SYMBOL TABLE

    BUFSIZE          21     106     160
    CDECL            24      25
    FALSE            28     102+    114     170
    FILE            105
    LINELEN          20     104     162
    NULL             34      40      69      93     105     155     157     162     165
    TRUE             28     103     118     121     125     159
    UCHAR_MAX        32      45
    _IOFBF          160
    _MSC_VER         23
    _QC              23
    _nullcheck               25
    _setenvp         24
    argc             98     108     132     134     153
    argv             98     108     110     129     131     146     157     169     173
    buffer          106     160
    c       104     108+
    ctype            17
    defined          23+
    fclose          156
    fgets           162
    find             96     150     165
    find_case        51      96
    find_no_case             72     150
    first           100     159     167     170
    fopen           157
    fprintf         129     136
    free             41
    h        13      14      15      16      17
    here             56      64      65      77      88      89
    i        38      45+     46      47+     48+
    infile          105     155     156     157     160     162
    init_find        36     152
    k       100     153+    157     169     173
    len              33      43      46      47      48      55      64+     76      88+
    limit            53      58      60      74      79      81
    limits           16
    line            101     161     164     175
    line_buffer             104     162     165     176
    line_numbers            102     121     174
    main             98
    memcmp           64
    memicmp          88
    new_string               36      42
    no_case         103     114     147
    pos              55      58      60+     61      64      66      76      79      81      82      84
            88      90
    print_file_name         102     118     167     172
    printf          169     173     175     176
    reverse         102     125     165
    setvbuf         160
    shift            54      60      61      75      82      84
    size_t           32      33      38      53      54      55      74      75      76
    stderr          129     136
    stdio            13
    stdlib           14
    strdup           42
    string           15      34      40      41      42      43      48      64      88     104     146
           149     152
    string2          51      53      60      64      72      74      82      88
    strlen           43      53      74
    strupr          149
    table            32      46      48      60      82
    tolower         110
    toupper          82
    uchar            30      60      82
```

```
/* +++Date last modified: 05-Jul-1997 */

/*----------------------------------------------------------------*/
/* Debugging extension by Jeff Dunlop                             */
/* Copyright 1992-1993, DB/Soft Publishing Co.                    */
/* License is hereby granted for use of JMalloc as a debugging    */
/* aid in any program.  JMalloc may not be sold or distributed    */
/* as or part of any for-profit debugging program or library      */
/* nor may it be included in any for-profit library that offers   */
/* debugging features.  Any redistribution of JMalloc source      */
/* must include this copyright notice.                            */
/*----------------------------------------------------------------*/

/*------------------------[ jmalloc.c ]-------------------------*/
/*                 drop-in for malloc with diags                  */
/*----------------------------------------------------------------*/

/*----------------------------------------------------------------*/
/*--------------------------[ notes ]-----------------------------*/
/*----------------------------------------------------------------*/

/*

The J... macros are intended for use on JMalloc'd memory blocks.  The
A... macros are intended for use on auto array blocks and will only
work if the base of the array is passed.  They will not work if the
address of an array element is passed.  All functions are designed to
be nearly drop-in replacements for standard library functions.  The one
case where debugging libraries traditionally fall short is when a block
operation occurs on an automatic array in a position other than at the
head.  This library could conceivably be extended to include a class of
functions that allow you to pass both a block and an offset into that
block so that the block and its size can be checked.  My style of
programming eschews this practice so I haven't had any motivation to so
extend this library.  Note that such an extension would depart slightly
from the standard syntax of the runtime library because of the
additional passed parameter.

JMemcheck(0) may be called at any time for an overwrite check of all
allocated blocks.  JMemcheck(1) additionally checks for orphaned blocks
and should be called just before program shutdown.

All functions are designed to log any detectable errors when the errors
occur, and to report either the line number of the overwrite, or the
line number that the block was allocated on, depending on which is more
useful.  Any blocks that are damaged by non-JMalloc functions will not
be noticed until they are JFree'd and will be tougher to debug.
Regardless of the damage that a function call produces, JMalloc never
departs from the runtime library's behavior.  Its job is to report,
nothing more.  All allocated blocks (except for JCalloc calls) are
dirtied and all blocks are dirtied before they are freed.  If you are
referencing freed memory or not initializing a new block, it will be
very obvious.  All blocks have an extra check byte that is checked when
the block is freed or during JMemcheck.  This doesn't catch random
memory writes, but it does catch overruns.

*/

/*----------------------------------------------------------------*/
/*-----------------------[ header files ]-------------------------*/
/*----------------------------------------------------------------*/

#define DBUG

#if defined __TURBOC__
 #include <alloc.h>
 #include <mem.h>
#else
 #if defined(__ZTC__)
  #include <dos.h>
 #else
  #include <malloc.h>
 #endif
#endif

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jmalloc.h"

/*----------------------------------------------------------------*/
/*--------------------[ local prototypes ]----------------------*/
/*----------------------------------------------------------------*/

static MLINK *JMemOwner(char *Addr);

/*----------------------------------------------------------------*/
/*---------------------[ global variables ]----------------------*/
/*----------------------------------------------------------------*/

MLINK *MalTop;                        /* top of allocation chain */

void db_prn(char *fmt, ...)
{
    va_list p;

    va_start(p, fmt);
    vprintf(fmt, p);
    putchar('\n');
```

```
    va_end(p);
}

void *j_deref(void *a, char *file, int line)
{
    if ( a == NULL )
    {
        DBUG_PRINT("alloc", ("Dereferenced NULL pointer - %s: line %d",
            file, line));
    }
    return a;
}

/*-------------------------[ j_malloc ]-------------------------*/
/*            Memory allocator with diagnostics                */
/*-------------------------------------------------------------*/
/* input:                                                      */
/*      Size = number of bytes to allocate                     */
/* local:                                                      */
/*      CurMal = pointer current mem struct                    */
/*      PrevMal = pointer to previous mem struct               */
/*      AllocAddr = pointer to allocated ram                   */
/* return:                                                     */
/*      address of allocated ram, or NULL on error             */
/* note:                                                       */
/*       use the JMalloc macro                                 */
/*-------------------------------------------------------------*/

static int memc;

void *j_malloc(unsigned Size, char *file, int line)
{
    /* 1. Allocate the memory */
    /* 2. Add to allocation chain */
    /* 3. Fill with non-null */

    MLINK *CurMal = MalTop,
          *PrevMal;

    void *AllocAddr;

    /* Allocate the memory + 1 (for check byte) */

    if ( (AllocAddr = malloc(Size + 1)) == NULL )
    {
        DBUG_PRINT("alloc", ("Allocation failed: %s Line %d", file, line));
        return((void*)0);
    }

    /* Add to the allocation chain */

    PrevMal = CurMal;
    while ( CurMal != NULL )
    {
        PrevMal = CurMal;
        CurMal = CurMal->NextLink;
    }
    if ( (CurMal = malloc(sizeof *CurMal)) == NULL )
    {
        DBUG_PRINT("alloc", ("Control allocation failed: %s Line %d", file,
            line));
        return(AllocAddr);
    }

    /* Deal with bootstrap */
    if ( PrevMal == NULL )
        MalTop = CurMal;
    else
        PrevMal->NextLink = CurMal;
    CurMal->NextLink = NULL;
    CurMal->MAddr = AllocAddr;
    CurMal->MSize = Size;
    CurMal->MLine = line;
    AStrCpy(CurMal->MFile, file);

    memset(AllocAddr, CKBYT, Size + 1);
    memc++;
    return(AllocAddr);
}

/*-------------------------[ j_calloc ]-------------------------*/
/*            Memory allocator with diagnostics                */
/*-------------------------------------------------------------*/
/* method:                                                     */
/*      - Allocate the memory via JMalloc                      */
/*      - Fill with null                                       */
/*      - Set check-byte                                       */
/* input:                                                      */
/*      Size = number of bytes to allocate                     */
/* local:                                                      */
/*      CurCal = pointer current mem struct                    */
/*      PrevCal = pointer to previous mem struct               */
/*      AllocAddr = pointer to allocated ram                   */
/* return:                                                     */
/*      address of allocated ram, or NULL on error             */
/*  note:                                                      */
/*      Use the JCalloc macro                                  */
/*-------------------------------------------------------------*/

void *j_calloc(unsigned Size, unsigned SizEach, char *file, int line)
```

```
{
    char *AllocAddr;

    /* Allocate the memory + 1 (for check byte) */

    /* Do not piss over NULL return - JMalloc handled that */
    if ( (AllocAddr = j_malloc(Size * SizEach, file, line)) != NULL )
    {
        /* Prep allocated ram */
        memset(AllocAddr, 0, Size * SizEach);
        AllocAddr[Size * SizEach] = CKBYT;
    }

    return(AllocAddr);
}

/*-------------------------[ j_realloc ]-------------------------*/
/*                     Reallocate memory                         */
/*---------------------------------------------------------------*/
/* input:                                                        */
/*  Addr = block's current address                               */
/*  Size = block's new size                                      */
/* return:                                                       */
/*  Block's new address, or NULL if memory not available.        */
/* note:                                                         */
/*  Use the JRealloc macro.                                      */
/*  If NULL is returned, Addr was also freed                     */
/*---------------------------------------------------------------*/

void *j_realloc(void *Addr, unsigned Size, char *file, int line)
{
    MLINK *CurMal = MalTop;

    void *tmp;

    if ( Addr == NULL )
        return(j_malloc(Size, file, line));
    /* Find the old block in the alloc list */
    while ( CurMal->MAddr != Addr && CurMal != NULL )
        CurMal = CurMal->NextLink;
    if ( CurMal == NULL )
    {
        /* Just call the standard realloc since we don't know anything
         * about this block
         */
        DBUG_PRINT("alloc", ("Realloc attempted on unrecorded block: %s "
            "Line %d", file, line));
        return(realloc(Addr, Size));
    }
    else
    {
        tmp = j_malloc(Size, file, line);
        if ( tmp != NULL )
        {
            memcpy(tmp, Addr, min(CurMal->MSize, Size));
            j_free(Addr, file, line);
        }
        else
        {
            j_free(Addr, file, line);
            return NULL;
        }
    }
    return(tmp);
}

/*-------------------------[ j_free ]--------------------------*/
/*           Memory deallocator with diagnostics               */
/*-------------------------------------------------------------*/
/* input:                                                      */
/*      AllocAddr = pointer to ram to deallocate               */
/* local:                                                      */
/*      CurMal = pointer to current mem struct                 */
/*      PrevMal = pointer to prev mem struct                   */
/* note:                                                       */
/*      This function is designed to be implemented with the   */
/*      macro JFree                                            */
/*-------------------------------------------------------------*/

void j_free(void *AllocAddr, char *file, int line)
{
    /* 1. Find the block in the alloc list */
    /* 2. Check for check-byte overwrite   */
    /* 3. Remove allocation record         */
    /* 3. Free the ram                     */

    MLINK *CurMal = MalTop,
          *PrevMal = MalTop;

    /* Find the block in the alloc list */
    while ( CurMal != NULL && CurMal->MAddr != AllocAddr )
    {
        PrevMal = CurMal;
        CurMal = CurMal->NextLink;
    }
    if ( CurMal == NULL )
    {
        DBUG_PRINT("alloc", ("Freeing an unrecorded block: %s %d", file,
            line));
    }
```

```
301 |     else
302 |     {
303 |         /* Check for check-byte overwrite */
304 |         if ( CurMal->MAddr[CurMal->MSize] != CKBYT )
305 |             DBUG_PRINT("alloc", ("Memory overrun detected on block allocated"
306 |             " at %s Line %d", CurMal->MFile, CurMal->MLine));
307 |         memset(AllocAddr, DIRTY, CurMal->MSize + 1);
308 |
309 |         if ( CurMal == MalTop )
310 |             MalTop = CurMal->NextLink;
311 |         else
312 |             PrevMal->NextLink = CurMal->NextLink;
313 |         free(CurMal);
314 |     }
315 |
316 |     /* Free the ram regardless of validity */
317 |     free(AllocAddr);
318 |     memc--;
319 |     return;
320 | }
321 |
322 | /*-------------------------[ JMemcheck ]-------------------------*/
323 | /*                Verify memory chain integrity                */
324 | /*-------------------------------------------------------------*/
325 | /* input:                                                      */
326 | /*  CheckFree != if currently allocated blocks are to be       */
327 | /*  reported                                                   */
328 | /* local:                                                      */
329 | /*      i = link number                                        */
330 | /*      CurMal = link pointer                                  */
331 | /*      status = current error condition                       */
332 | /* return:                                                     */
333 | /*      -1 = error,                                            */
334 | /*      0 = no error detected                                  */
335 | /*-------------------------------------------------------------*/
336 |
337 | int j_memcheck(int CheckFree)
338 | {
339 |     int status = 0;
340 |
341 |     MLINK *CurMal = MalTop;
342 |
343 |     /* Walk the alloc list */
344 |     if ( CheckFree >= 2 )
345 |     {
346 |         DBUG_PRINT("alloc", ("There should be %d unfreed blocks", memc));
347 |     }
348 |
349 |     while ( CurMal != NULL )
350 |     {
351 |         if ( CurMal->MAddr[CurMal->MSize] != CKBYT )
352 |         {
353 |             DBUG_PRINT("alloc", ("Memory overrun detected in link "
354 |             "allocated at %s Line %d", CurMal->MFile, CurMal->MLine));
355 |             status = -1;
356 |         }
357 |         if ( CheckFree >= 1 )
358 |         {
359 |             DBUG_PRINT("alloc", ("Unfreed block of size %d allocated at "
360 |             "%s Line %d", CurMal->MSize, CurMal->MFile, CurMal->MLine));
361 |             status = -1;
362 |         }
363 |         CurMal = CurMal->NextLink;
364 |     }
365 |     return(status);
366 | }
367 |
368 | /*-------------------------[ JMemValid ]-------------------------*/
369 | /*          Verify an address is in a JMalloc space            */
370 | /*-------------------------------------------------------------*/
371 |
372 | int JMemValid(char *Addr)
373 | {
374 |     MLINK *CurMal = MalTop;
375 |
376 |     while ( CurMal != NULL )
377 |     {
378 |         if ( Addr >= CurMal->MAddr && Addr <= CurMal->MAddr + CurMal->MSize )
379 |             return(TRUE);
380 |         CurMal = CurMal->NextLink;
381 |     }
382 |
383 |     return(FALSE);
384 | }
385 |
386 | /*-------------------------[ JMemOwner ]-------------------------*/
387 | /*            Return the owner of a block of ram               */
388 | /*-------------------------------------------------------------*/
389 |
390 | static MLINK *JMemOwner(char *Addr)
391 | {
392 |     MLINK *CurMal = MalTop;
393 |
394 |     while ( CurMal != NULL )
395 |     {
396 |         unsigned long Cur = (unsigned long)CurMal->MAddr,
397 |                       Adr = (unsigned long)Addr;
398 |         if ( Adr >= Cur && Adr <= Cur + CurMal->MSize )
399 |             return(CurMal);
400 |         CurMal = CurMal->NextLink;
```

```
401 |     }
402 |
403 |     return((void*)NULL);
404 | }
405 |
406 | /*-------------------------[ j_strcpy ]-------------------------*/
407 | /*          String copy with checks and protections             */
408 | /*-------------------------------------------------------------*/
409 |
410 | char *j_strcpy(char *__dest, const char *__src, char *file, int line)
411 | {
412 |     MLINK *MemOwner = JMemOwner(__dest);
413 |
414 |     if ( MemOwner == NULL )
415 |     {
416 |         DBUG_PRINT("alloc", ("Strcpy destination invalid - %s: line %d",
417 |             file, line));
418 |     }
419 |     else if ( strlen(__src) > MemOwner->MAddr + MemOwner->MSize - __dest )
420 |     {
421 |         DBUG_PRINT("alloc", ("Strcpy destination overrun - %s: line %d",
422 |             file, line));
423 |     }
424 |     return(strcpy(__dest, __src));
425 | }
426 |
427 | /*-------------------------[ j_strncpy ]------------------------*/
428 | /*                String n copy with checks                    */
429 | /*-------------------------------------------------------------*/
430 |
431 | char *j_strncpy(char *__dest, const char *__src, size_t maxlen, char *file,
432 |                 int line)
433 | {
434 |     MLINK *MemOwner = JMemOwner(__dest);
435 |
436 |     if ( MemOwner == NULL )
437 |     {
438 |         DBUG_PRINT("alloc", ("strncpy destination invalid - %s: line %d",
439 |             file, line));
440 |     }
441 |     else if ( maxlen > MemOwner->MAddr + MemOwner->MSize - __dest )
442 |     {
443 |         DBUG_PRINT("alloc", ("strncpy destination overrun - %s: line %d",
444 |             file, line));
445 |     }
446 |
447 |     return(strncpy(__dest, __src, maxlen));
448 | }
449 |
450 | char *j_strcat(char *__dest, char *__src, char *file, int line)
451 | {
452 |     MLINK *MemOwner = JMemOwner(__dest);
453 |
454 |     if ( MemOwner == NULL )
455 |     {
456 |         DBUG_PRINT("alloc", ("strcat destination invalid - %s: line %d",
457 |             file, line));
458 |     }
459 |     else if ( strlen(MemOwner->MAddr) + strlen(__src) + 1 > MemOwner->MSize )
460 |     {
461 |         DBUG_PRINT("alloc", ("strcat destination overrun - %s: line %d",
462 |             file, line));
463 |     }
464 |
465 |     return(strcat(__dest, __src));
466 | }
467 |
468 | char *j_strdup(char *__str, char *file, int line)
469 | {
470 |     char *p = j_malloc(strlen(__str) + 1, file, line);
471 |
472 |     if ( p != NULL )
473 |     {
474 |         j_strcpy(p, __str, file, line);
475 |     }
476 |     return(p);
477 | }
478 |
479 | char *j_strnset(char *str, int ch, size_t n, char *file, int line)
480 | {
481 |     MLINK *MemOwner = JMemOwner(str);
482 |
483 |     if ( MemOwner == NULL )
484 |     {
485 |         DBUG_PRINT("alloc", ("strnset destination invalid - %s: line %d",
486 |             file, line));
487 |     }
488 |     return(strnset(str, ch, n));
489 | }
490 |
491 | char *j_strset(char *str, int ch, char *file, int line)
492 | {
493 |     MLINK *MemOwner = JMemOwner(str);
494 |
495 |     if ( MemOwner == NULL )
496 |     {
497 |         DBUG_PRINT("alloc", ("strset destination invalid - %s: line %d",
498 |             file, line));
499 |     }
500 |     return(strset(str, ch));
```

```
501 | }
502 |
503 | /*------------------------[ j_checkstr ]------------------------*/
504 | /*             Check a MLINK string for problems                */
505 | /* j_sprintf      sprintf replacement                            */
506 | /*-----------------------------------------------------------------*/
507 |
508 | int j_checkstr(MLINK *str)
509 | {
510 |     if ( str->MAddr[str->MSize] != CKBYT || strlen(str->MAddr) > str->MSize - 1 )
511 |     {
512 |         DBUG_PRINT("alloc",
513 |             ("Bad string, allocated at %s Line %d", str->MFile, str->MLine));
514 |         return 1;
515 |     }
516 |     return 0;
517 | }
518 |
519 | void *j_memcpy(void *dest, const void *src, size_t n, char *file, int line)
520 | {
521 |     MLINK *MemOwner = JMemOwner(dest);
522 |
523 |     if ( MemOwner == NULL )
524 |     {
525 |         DBUG_PRINT("alloc", ("memcpy destination invalid - %s: line %d",
526 |             file, line));
527 |     }
528 |     else if ( n > MemOwner->MAddr + MemOwner->MSize - (char *)dest )
529 |     {
530 |         DBUG_PRINT("alloc", ("memcpy destination overrun - %s: line %d",
531 |             file, line));
532 |     }
533 |
534 |     return(memcpy(dest, src, n));
535 | }
536 |
537 | void *j_memset(void *dest, int ch, size_t n, char *file, int line)
538 | {
539 |     MLINK *MemOwner = JMemOwner(dest);
540 |
541 |     if ( MemOwner == NULL )
542 |     {
543 |         DBUG_PRINT("alloc", ("memset destination invalid - %s: line %d",
544 |             file, line));
545 |     }
546 |     else if ( n > MemOwner->MAddr + MemOwner->MSize - (char *)dest )
547 |     {
548 |         DBUG_PRINT("alloc", ("memset destination overrun - %s: line %d",
549 |             file, line));
550 |     }
551 |
552 |     return(memset(dest, ch, n));
553 | }
554 |
555 | char *a_strcpy(char *__dest, const char *__src, int size, char *file,
556 |     int line)
557 | {
558 |     if ( strlen(__src) >= size)
559 |         DBUG_PRINT("alloc", ("strcpy destination overrun - %s: line %d",
560 |             file, line));
561 |     return(strcpy(__dest, __src));
562 | }
563 |
564 | char *a_strncpy(char *dest, const char *src, size_t n, int size, char *file,
565 |     int line)
566 | {
567 |     if ( n >= size )
568 |         DBUG_PRINT("alloc", ("strncpy destination overrun - %s: line %d",
569 |             file, line));
570 |     return(strncpy(dest, src, n));
571 | }
572 |
573 | char *a_strcat(char *dest, const char *src, int size, char *file, int line)
574 | {
575 |     if ( strlen(dest) + strlen(src) + 1 > size )
576 |         DBUG_PRINT("alloc", ("strcat destination overrun = %s: line %d",
577 |             file, line));
578 |     return(strcat(dest, src));
579 | }
580 |
581 | char *a_strnset(char *str, int ch, size_t n, int size, char *file, int line)
582 | {
583 |     if ( n + 1 > size )
584 |         DBUG_PRINT("alloc", ("strnset destination overrun - %s: line %d",
585 |             file, line));
586 |     return(strnset(str, ch, n));
587 | }
588 |
589 | void *a_memcpy(void *dest, const void *src, size_t n, int size, char *file,
590 |              int line)
591 | {
592 |     if ( n > size )
593 |         DBUG_PRINT("alloc", ("memcpy destination overrun - %s: line %d",
594 |             file, line));
595 |     return(memcpy(dest, src, n));
596 | }
597 |
598 | void *a_memset(void *dest, int ch, size_t n, int size, char *file, int line)
599 | {
600 |     if ( n > size )
```

```
601 |          DBUG_PRINT("alloc", ("memset destination overrun - %s: line %d",
602 |              file, line));
603 |      return memset(dest, ch, n);
604 | }
605 |
606 | #ifdef TEST
607 |
608 | #ifdef __WATCOMC__
609 |  #pragma off (unreferenced);
610 | #endif
611 | #ifdef __TURBOC__
612 |  #pragma argsused
613 | #endif
614 |
615 | int main(int argc, char *argv[])
616 | {
617 |     char *new,
618 |          *p,
619 |          *q,
620 |          r[30],
621 |          *s[256] = {NULL};
622 |     int i = 1;
623 |
624 |     while ( (p = argv[i++]) != NULL )
625 |     {
626 |         switch ( *p++ )
627 |             case '-':
628 |                 switch ( *p++ )
629 |                     case '#':
630 |                         DBUG_PUSH(argv[i - 1]);
631 |     }
632 |     new = JCalloc(64, 56);
633 |     new = JRealloc(new, 64);
634 |     JStrCpy(new + 10, "Test string");
635 |     DBUG_PRINT("alloc", ("r is invalid"));
636 |     JStrCpy(r, "Test");
637 |     q = JCalloc(4, 4);
638 |     q[4 * 4] = 3;
639 |
640 |     DBUG_PRINT("alloc", ("p was never allocated"));
641 |     JFree(p);
642 |
643 |     i = 0;
644 |
645 |     DBUG_PRINT("alloc", ("Deplete memory"));
646 |     do
647 |     {
648 |         s[i] = JMalloc(32000);
649 |         i++;
650 |     } while ( s[i - 1] != NULL );
651 |
652 |     i = 0;
653 |     while ( s[i] != NULL )
654 |     {
655 |         JFree(s[i++]);
656 |     }
657 |
658 |     DBUG_PRINT("alloc", ("New and q are orphaned blocks"));
659 |     DBUG_PRINT("alloc", ("q has an overrun"));
660 |     JMemcheck(1);
661 |     return(0);
662 | }
663 | #endif
```

TEXT STATISTICS

```
 21990 characters
   663 lines
```

LEXICAL STATISTICS

```
   133 comments [std-C]
     0 comments [C++]
    24 preprocessor instructions
     3 constants [character]
    73 constants [string]
    36 constants [numeric]
```

SYMBOL TABLE

```
    AStrCpy         174
    Addr             86     230     236     239     248     255     256     260     372     378+    390
           397
    Adr             397     398+
    AllocAddr       140     144     162     171     176     178     202     207     210     211     214
           280      291     307     317
    CKBYT           176     211     304     351     510
    CheckFree       337     344     357
    Cur             396     398+
    CurMal          137     152     153     155     156+    158+    167     169     170     171     172
           173      174     232     239+    240+    241     255     287     291+    293     294+    296
           304+     306+    307     309     310     312     313     341     349     351+    354+    360+
           363+     374     376     378+    380+    392     394     396     398     399     400+
    DBUG             63
    DBUG_PRINT              108     146     160     246     298     305     346     353     359     416
           421      438     443     456     461     485     497     512     525     530     543     548
           559      568     576     584     593     601     635     640     645     658     659
    DBUG_PUSH       630
    DIRTY           307
    FALSE           383
    JCalloc         632     637
    JFree           641     655
    JMalloc         648
    JMemOwner        86     390     412     434     452     481     493     521     539
    JMemValid       372
    JMemcheck       660
    JRealloc        633
    JStrCpy         634     636
    MAddr           171     239     291     304     351     378+    396     419     441     459     510+
           528      546
    MFile           174     306     354     360     513
    MLINK            86      92     137     232     287     341     374     390     392     412     434
           452      481     493     508     521     539
    MLine           173     306     354     360     513
    MSize           172     255     304     307     351     360     378     398     419     441     459
           510+     528     546
    MalTop           92     137     167     232     287     288     309     310     341     374     392
    MemOwner        412     414     419+    434     436     441+    452     454     459+    481     483
           493      495     521     523     528+    539     541     546+
    NULL            106     144     153     158     166     170     207     236     239     241     253
           261      291     296     349     376     394     403     414     436     454     472     483
           495      523     541     621     624     650     653
    NextLink        156     169     170     240     294     310     312+    363     380     400
    PrevMal         138     152     155     166     169     288     293     312
    SizEach         200     207     210     211
    Size            131     144     172     176     200     207     210     211     230     237     248
           252      255
    TEST            606
    TRUE            379
    __TURBOC__               65     611
    __WATCOMC__             608
    __ZTC__          69
    __dest          410     412     419     424     431     434     441     447     450     452     465
           555      561
    __src           410     419     424     431     447     450     459     465     555     558     561
    __str           468     470     474
    a       104     106     111
    a_memcpy        589
    a_memset        598
    a_strcat        573
    a_strcpy        555
    a_strncpy       564
    a_strnset       581
    alloc            66
    argc            615
    argsused        612
    argv            615     624     630
    ch              479     488     491     500     537     552     581     586     598     603
    db_prn           94
    defined          65      69
    dest            519     521     528     534     537     539     546     552     564     570     573
           575      578     589     595     598     603
    dos              70
    file            104     109     131     146     160     174     200     207     230     237     247
           252      256     260     280     298     410     417     422     431     439     444     450
           457      462     468     470     474     479     486     491     498     519     526     531
           537      544     549     555     560     564     569     573     577     581     585     589
           594      598     602
    fmt              94      98      99
    free            313     317
    h        66      67      70      72      76      77      78      79
    i       622     624     630     643     648     649     650     652     653     655
    j_calloc        200
    j_checkstr              508
```

```
j_deref          104
j_free           256     260     280
j_malloc         131     207     237     252     470
j_memcheck               337
j_memcpy         519
j_memset         537
j_realloc        230
j_strcat         450
j_strcpy         410     474
j_strdup         468
j_strncpy        431
j_strnset        479
j_strset         491
line             104     109     131     146     161     173     200     207     230     237     247
       252       256     260     280     299     410     417     422     432     439     444     450
       457       462     468     470     474     479     486     491     498     519     526     531
       537       544     549     556     560     565     569     573     577     581     585     590
       594       598     602
main             615
malloc            72     144     158
maxlen           431     441     447
mem               67
memc             129     177     318     346
memcpy           255     534     595
memset           176     210     307     552     603
min              255
n      479       488     519     528     534     537     546     552     564     567     570     581
       583       586     589     592     595     598     600     603
off              609
p       96        98      99     101     470     472     474     476     618     624     626     628
       641
putchar          100
q      619       637     638
r      620       636
realloc          248
s      621       648     650     653     655
size             555     558     564     567     573     575     581     583     589     592     598
       600
size_t           431     479     519     537     564     581     589     598
src              519     534     564     570     573     575     578     589     595
status           339     355     361     365
stdarg            76
stdio             77
stdlib            78
str              479     481     488     491     493     500     508     510+    513+    581     586
strcat           465     578
strcpy           424     561
string            79
strlen           419     459+    470     510     558     575+
strncpy          447     570
strnset          488     586
strset           500
tmp              234     252     253     255     264
unreferenced             609
va_end           101
va_list           96
va_start          98
vprintf           99
```

```
/* +++Date last modified: 05-Jul-1997 */

/*----------------------------------------------------------------*/
/* Debugging extension by Jeff Dunlop                             */
/* Copyright 1992-1993, DB/Soft Publishing Co.                    */
/* License is hereby granted for use of JMalloc as a debugging    */
/* aid in any program.  JMalloc may not be sold or distributed    */
/* as or part of any for-profit debugging program or library      */
/* nor may it be included in any for-profit library that offers   */
/* debugging features.  Any redistribution of JMalloc source      */
/* must include this copyright notice.                            */
/*----------------------------------------------------------------*/

/*-----------------------[ jmalloc.h ]----------------------------*/
/*                  main header file for jmalloc                  */
/*----------------------------------------------------------------*/

/*----------------------------------------------------------------*/
/*--------------------------[ defines ]---------------------------*/
/*----------------------------------------------------------------*/
#ifndef _jmalloc
#   define _jmalloc

#define FALSE 0
#define TRUE 1

#define CKBYT 1
#define DIRTY 2

/*----------------------------------------------------------------*/
/*-----------------------[ structures ]---------------------------*/
/*----------------------------------------------------------------*/

typedef struct mlist
{
    struct mlist *NextLink;         /* link to next struct */
    char *MAddr;                    /* address assigned    */
    unsigned MSize;                 /* size allocated      */
    char MFile[20];                 /* Allocation file name */
    int MLine;                      /* Allocation line number */
} MLINK;

/*----------------------------------------------------------------*/
/*--------------------[ public prototypes ]----------------------*/
/*----------------------------------------------------------------*/

#ifdef __cplusplus
   extern   "C"   {
#endif

void db_prn(char *fmt, ...);
#define DBUG_ENTER(a)
#define DBUG_RETURN(a) return(a)
#define DBUG_PRINT(a, b) db_prn b
#define DBUG_PUSH(a)
#define DBUG_VOID_RETURN return

#define JCheckStr(a) j_checkstr((a))

int j_checkstr(MLINK *str);

#ifdef DBUG

#define dr(a) j_deref(a, __FILE__, __LINE__)

#define JStrnSet(a, b, c) j_strnset((a), (b), (c), __FILE__, __LINE__)
#define JStrDup(a) j_strdup((a), __FILE__, __LINE__)
#define JStrCat(a, b) j_strcat((a), (b), __FILE__, __LINE__)
#define JMalloc(a) j_malloc((a), __FILE__, __LINE__)
#define JCalloc(a, b) j_calloc((a), (b), __FILE__, __LINE__)
#define AStrCpy(a, b) a_strcpy((a), (b), sizeof (a), __FILE__, __LINE__)
#define AStrCat(a, b) a_strcat((a), (b), sizeof (a), __FILE__, __LINE__)
#define AStrnCpy(a, b, c) a_strncpy((a), (b), (c), sizeof (a), __FILE__, __LINE__)
#define AStrnSet(a, b, c) a_strnset((a), (b), (c), sizeof (a), __FILE__, __LINE__)
#define JFree(a) j_free(a, __FILE__, __LINE__)
#define JStrCpy(a, b) j_strcpy((a), (b), __FILE__, __LINE__)
#define JStrnCpy(a, b, c) j_strncpy((a), (b), (c), __FILE__, __LINE__)
#define JMemCpy(a, b, c) j_memcpy((a), (b), (c), __FILE__, __LINE__)
#define JMemSet(a, b, c) j_memset((a), (b), (c), __FILE__, __LINE__)
#define JMemcheck(a) j_memcheck((a))
#define AMemCpy(a, b, c) a_memcpy((a), (b), (c), sizeof (a), __FILE__, __LINE__)
#define AMemSet(a, b, c) a_memset((a), (b), (c), sizeof (a), __FILE__, __LINE__)
#define JRealloc(a, b) j_realloc((a), (b), __FILE__, __LINE__)

void *j_deref(void *a, char *file, int line);
char *j_strnset(char *str, int ch, size_t n, char *file, int line);
char *j_strdup(char *str, char *file, int line);
char *j_strcat(char *__dest, char *__src, char *file, int line);
void *j_malloc(unsigned size, char *file, int line);
void *j_calloc(unsigned size, unsigned sizeach, char *file, int line);
int j_memcheck(int CheckFree);
void j_free(void *AllocAddr, char *file, int line);
char *j_strcpy(char *__dest, const char *__src, char *file, int line);
char *j_strncpy(char *__dest, const char *__src, size_t maxlen, char *file,
                int line);
void *j_realloc(void *addr, unsigned Size, char *file, int line);
void *j_memset(void *dest, int ch, size_t n, char *file, int line);
void *j_memcpy(void *dest, const void *src, size_t n, char *file, int line);
char *a_strcpy(char *__dest, const char *__src, int size, char *file,
               int line);
```

```
101 | char*a_strcat(char *dest, const char *src, int size, char *file, int line);
102 | void *a_memcpy(void *dest, const void *src, size_t n, int size, char *file,
103 |                int line);
104 | void *a_memset(void *dest, int ch, size_t n, int size, char *file, int line);
105 | char *a_strncpy(char *__dest, const char *__src, size_t maxlen, int size,
106 |     char *file, int line);
107 | char *a_strnset(char *str, int ch, size_t n, int size, char *file, int line);
108 |
109 |
110 | #else
111 |
112 | #define dr(a) (a)
113 |
114 | #define JStrnSet(a, b, c) strnset((a), (b), (c))
115 | #define JStrDup(a) strdup((a))
116 | #define JStrCat(a, b) strcat((a), (b))
117 | #define JMalloc(a) malloc((a))
118 | #define JCalloc(a, b) calloc((a), (b))
119 | #define JFree(a) free(a)
120 | #define JStrCpy(a, b) strcpy((a), (b))
121 | #define JStrnCpy(a, b, c) strcpy((a), (b), (c))
122 | #define JMemCpy(a, b, c) memcpy((a), (b), (c))
123 | #define JMemSet(a, b, c) memset((a), (b), (c))
124 | #define JMemcheck(a)
125 | #define JRealloc(a, b) realloc((a), (b))
126 |
127 | #define AStrCpy(a, b) strcpy((a), (b))
128 | #define AStrCat(a, b) strcat((a), (b))
129 | #define AStrnCpy(a, b, c)  strncpy((a), (b), (c))
130 | #define AStrnSet(a, b, c), strnset((a), (b), (c))
131 | #define AMemCpy(a, b, c) memcpy((a), (b), (c))
132 | #define AMemSet(a, b, c) memset((a), (b), (c))
133 |
134 | #endif
135 |
136 | #ifdef __cplusplus
137 |    }
138 | #endif
139 |
140 | /*---------------------------------------------------------------*/
141 | /*--------------------[ public variables ]---------------------*/
142 | /*---------------------------------------------------------------*/
143 |
144 | extern MLINK *MalTop;
145 | #endif
```

```
TEXT STATISTICS

  6039 characters
   145 lines

LEXICAL STATISTICS

    31 comments [std-C]
     0 comments [C++]
    57 preprocessor instructions
     0 constants [character]
     1 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    AMemCpy          81     131
    AMemSet          82     132
    AStrCat          72     128
    AStrCpy          71     127
    AStrnCpy         73     129
    AStrnSet         74     130
    AllocAddr        92
    CKBYT            27
    CheckFree        91
    DBUG             62
    DBUG_ENTER               52
    DBUG_PRINT               54
    DBUG_PUSH        55
    DBUG_RETURN              53
    DBUG_VOID_RETURN         56
    DIRTY            28
    FALSE            24
    JCalloc          70     118
    JCheckStr        58
    JFree            75     119
    JMalloc          69     117
    JMemCpy          78     122
    JMemSet          79     123
    JMemcheck        80     124
    JRealloc         83     125
    JStrCat          68     116
    JStrCpy          76     120
    JStrDup          67     115
    JStrnCpy         77     121
    JStrnSet         66     114
    MAddr            37
    MFile            39
    MLINK            41      60     144
    MLine            40
    MSize            38
    MalTop          144
    NextLink         36
    Size             96
    TRUE             25
    __FILE__         64      66      67      68      69      70      71      72      73      74      75
            76       77      78      79      81      82      83
    __LINE__         64      66      67      68      69      70      71      72      73      74      75
            76       77      78      79      81      82      83
    __cplusplus              47     136
    __dest           88      93      94      99     105
    __src            88      93      94      99     105
    _jmalloc         21      22
    a        52      53+     54      55      58+     64+     66+     67+     68+     69+     70+     71+
             72+     73+     74+     75+     76+     77+     78+     79+     80+     81+     82+     83+
             85     112+    114+    115+    116+    117+    118+    119+    120+    121+    122+    123+
            124     125+    127+    128+    129+    130+    131+    132+
    a_memcpy         81     102
    a_memset         82     104
    a_strcat         72     101
    a_strcpy         71      99
    a_strncpy        73     105
    a_strnset        74     107
    addr             96
    b        54+     66+     68+     70+     71+     72+     73+     74+     76+     77+     78+     79+
             81+     82+     83+    114+    116+    118+    120+    121+    122+    123+    125+    127+
            128+    129+    130+    131+    132+
    c        66+     73+     74+     77+     78+     79+     81+     82+    114+    121+    122+    123+
            129+    130+    131+    132+
    calloc           118
    ch               86      97     104     107
    db_prn           51      54
    define           22
    dest             97      98     101     102     104
    dr               64     112
    file             85      86      87      88      89      90      92      93      94      96      97
            98       99     101     102     104     106     107
    fmt              51
    free            119
    j_calloc         70      90
    j_checkstr               58      60
    j_deref          64      85
    j_free           75      92
    j_malloc         69      89
    j_memcheck               80      91
    j_memcpy         78      98
    j_memset         79      97
    j_realloc        83      96
    j_strcat         68      88
    j_strcpy         76      93
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*------------------------------------------------------------*/
  4 | /* Debugging extension by Jeff Dunlop                         */
  5 | /* Copyright 1992-1993, DB/Soft Publishing Co.                */
  6 | /* License is hereby granted for use of JMalloc as a debugging  */
  7 | /* aid in any program.  JMalloc may not be sold or distributed  */
  8 | /* as or part of any for-profit debugging program or library    */
  9 | /* nor may it be included in any for-profit library that offers */
 10 | /* debugging features.  Any redistribution of JMalloc source    */
 11 | /* must include this copyright notice.                        */
 12 | /*------------------------------------------------------------*/
 13 |
 14 | #include <stdlib.h>
 15 |
 16 | #ifndef jnew_h
 17 | #define jnew_h
 18 |
 19 | #ifdef DBUG
 20 |
 21 | #define NEW(a) (db_set(__FILE__, __LINE__), new a)
 22 | #define DELETE(a) (db_set(__FILE__, __LINE__), delete a)
 23 |
 24 | void *operator new(size_t n);
 25 | void operator delete(void *p);
 26 | void db_set(char *file, int line);
 27 |
 28 | #else
 29 |
 30 | #define NEW(a) new a
 31 | #define DELETE(a) delete a
 32 |
 33 | #endif /* DBUG */
 34 | #endif /* jnew_h */
```

TEXT STATISTICS

```
  1112 characters
    34 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DBUG            19
    DELETE          22      31
    NEW             21      30
    __FILE__        21      22
    __LINE__        21      22
    a       21+     22+     30+     31+
    db_set          21      22      26
    file            26
    h       14
    jnew_h          16      17
    line            26
    n       24
    p       25
    size_t          24
    stdlib          14
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  JOYSTICK.C
 5 | **
 6 | **  Joystick support functions
 7 | **
 8 | **  Public domain demo by Bob Stout
 9 | */
10 |
11 | #include <dos.h>
12 | #include "snpdosys.h"
13 |
14 | struct joystick JoyStick;
15 |
16 | /*
17 | **  read_joystick()
18 | **
19 | **  returns Success_ or Error_
20 | **
21 | **  fills in global JoyStick structure
22 | */
23 |
24 | Boolean_T read_joystick(void)
25 | {
26 |         union REGS regs;
27 |
28 |         regs.h.ah = 0x84;                       /* Read the switches    */
29 |         regs.x.dx = 0;
30 |         int86(0x15, &regs, &regs);
31 |         if (regs.x.cflag)
32 |                 return Error_;
33 |         JoyStick.switch_0 = TOBOOL(regs.h.al & 0x10);
34 |         JoyStick.switch_1 = TOBOOL(regs.h.al & 0x20);
35 |         JoyStick.switch_2 = TOBOOL(regs.h.al & 0x40);
36 |         JoyStick.switch_3 = TOBOOL(regs.h.al & 0x80);
37 |
38 |         regs.h.ah = 0x84;                       /* Read positions       */
39 |         regs.x.dx = 1;
40 |         int86(0x15, &regs, &regs);
41 |         if (regs.x.cflag)
42 |                 return Error_;
43 |         JoyStick.pos_Ax = regs.x.ax;
44 |         JoyStick.pos_Ay = regs.x.bx;
45 |         JoyStick.pos_Bx = regs.x.cx;
46 |         JoyStick.pos_By = regs.x.dx;
47 |
48 |         return Success_;
49 | }
```

TEXT STATISTICS

```
  1187 characters
    49 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       24
    Error_          32      42
    JoyStick        14      33      34      35      36      43      44      45      46
    REGS            26
    Success_        48
    TOBOOL          33      34      35      36
    ah              28      38
    al              33      34      35      36
    ax              43
    bx              44
    cflag           31      41
    cx              45
    dos             11
    dx              29      39      46
    h       11      28      33      34      35      36      38
    int86           30      40
    joystick        14
    pos_Ax          43
    pos_Ay          44
    pos_Bx          45
    pos_By          46
    read_joystick           24
    regs            26      28      29      30+     31      33      34      35      36      38      39
            40+     41      43      44      45      46
    switch_0        33
    switch_1        34
    switch_2        35
    switch_3        36
    x       29      31      39      41      43      44      45      46
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  KBFLIP.C
**
**  a public domain demo by: Bob Stout
*/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <process.h>

#ifdef __TURBOC__
 #define FAR far
#else
 #define FAR _far
#endif

#define SHOW(str) fputs(str"\n", stderr)

#define BitSet(arg,posn) ((arg) | (1L << (posn)))
#define BitClr(arg,posn) ((arg) & ~(1L << (posn)))

#define LOCKS_POSN 4
#define BYTE unsigned char

BYTE FAR *kb_status = (BYTE FAR *) 0x00400017L;

/*
**  Tell the folks how this works
*/

void usage(void)
{
      SHOW("Usage: KBFLIP {+|-}[switches] [...{+|-}[switches]]");
      SHOW("Where \"switches\" are one or more of:");
      SHOW(" +/-C - Turn Caps Lock on/off");
      SHOW(" +/-N - Turn Num Lock on/off");
      SHOW(" +/-S - Turn Scroll Lock on/off");
      SHOW("Note switches may be upper or lower case\n");
      SHOW("Example: \"KBFLIP +Cn -S\" turns Caps Lock and Num Lock on "
            "and Scroll lock off");
      exit(-1);
}

/*
**  The real works starts here
**
**  This works by checking the user input against a string containing the
**  allowable switch characters in the same relative positions they
**  occupy in the BIOS data area, offset by 4 (LOCKS_POSN).
**
**  Note that all changes are made to a copy of the BIOS data so any
**  input errors will not cause incomplete changes to be applied.
*/

int main(int argc, char *argv[])
{
      int i, j;
      char *args = "SNC";
      BYTE template = *kb_status;             /* Make changes to copy */

      if (2 > argc)                            /* Help 'em             */
            usage();
      for (i = 1; i < argc; ++i)
      {
            if (NULL == strchr("+-", *argv[i]))
                  usage();

            for (j = 1; argv[i][j]; ++j)
            {
                  char *found;

                  if (NULL != (found = strchr(args, toupper(argv[i][j]))))
                  {
                        int posn = LOCKS_POSN + (found - args);

                        if ('+' == *argv[i])
                              template = (BYTE)BitSet(template, posn);
                        else  template = (BYTE)BitClr(template, posn);
                  }
                  else  usage();
            }
      }
      *kb_status = template;                   /* Apply all changes    */
      return(0);
}
```

TEXT STATISTICS

2432 characters
88 lines

LEXICAL STATISTICS

7 comments [std-C]
0 comments [C++]
14 preprocessor instructions
1 constants [character]
10 constants [string]
9 constants [numeric]

SYMBOL TABLE

BYTE	26	28+	62	80	81			
BitClr	23	81						
BitSet	22	80						
FAR	15	17	28+					
LOCKS_POSN		25	77					
NULL	68	75						
SHOW	20	36	37	38	39	40	41	42
__TURBOC__		14						
_far	17							
arg	22+	23+						
argc	58	64	66					
args	61	75	77					
argv	58	68	71	75	79			
ctype	11							
exit	44							
far	15							
found	73	75	77					
fputs	20							
h 9	10	11	12					
i 60	66+	68	71	75	79			
j 60	71+	75						
kb_status	28	62	86					
main	58							
posn	22+	23+	77	80	81			
process	12							
stderr	20							
stdio	9							
str	20							
str"\n"	20							
strchr	68	75						
string	10							
toupper	75							
usage	34	65	69	83				

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  KBTRAP.C - Traps Ctrl-Alt-Del, Ctrl-C, Ctrl-Break, SysRq, PrintScreen,
**             and Pause keys.
**
**  Derived from public domain sources, uses SNIPPETS header files
*/

#include <stdlib.h>
#include <dos.h>
#include "pchwio.h"
#include "sniptype.h"
#include "sniprint.h"

#define CTRLALT         (0x08|0x04) /* bit flags set in kbstat()        */
#define CTRL            (0x04)      /* bit flags set in kbstat()        */
#define ALT             (0x08)      /* bit flags set in kbstat()        */
#define DELSCAN         0x53        /* keyboard scan code for <Del>     */
#define CSCAN           0x2E        /* keyboard scan code for <C>       */
#define BREAKSCAN       0x46        /* keyboard scan code for <Break>   */
#define PRTSCRNSCAN     0x37        /* keyboard scan code for <PrtScrn> */
#define SYSREQSCAN      0x54        /* keyboard scan code for <SysReq>  */
#define MULTI2SCAN      0xE0        /* 1st byte of 2 byte scan code     */
#define MULTI3SCAN      0xE1        /* 1st byte of 3 byte scan code     */
#define KEYPORT         0x60        /* keyboard scan code port          */
#define CONTROLLERPORT  0x20        /* interrupt controller port        */
#define kbstat()        Peekw(0,0x417) /* BIOS data area - kb flags     */

#define keyport()      inp(KEYPORT)
            /* macro that returns the scancode of the key that caused   */
            /* the interrupt                                            */

#define install()      (oldkbisr=getvect(0x09),setvect(0x09,newkbisr))
            /* installation macro, installs newkbisr() in the keyboard  */
            /* interrupt chain                                          */

#define uninstall()       setvect(0x09,oldkbisr)
            /* removal macro, call to remove newkbisr() from interrupt  */
            /* chain.  oldkbisr()  must be removed before program ends  */

static void (INTERRUPT FAR * oldkbisr)(void);
            /* address of old keyboard ISR                              */

static Boolean_T PrtScrnPending = False_;
            /* semaphore to flag pending print screen operations        */

static void cleanup(void);
            /* atexit()-registered version of uninstall() macro         */

static Boolean_T KBtrap_active;
static Boolean_T KBtrap_installed = False_;

static void KBtrap(void);
static void cleanup(void);
static void INTERRUPT FAR newkbisr(void);


/*
**  Called to activate trapping the system keys. When called the first time,
**  it calls KBtrap() to install the ISR. Registers the ISR removal function
**  using atexit() so the ISR will always be uninstalled upon program
**  termination.
*/

void activate_KBtrap(void)
{
      if (!KBtrap_installed)
            KBtrap();
      KBtrap_active = True_;
#ifdef TEST
      puts("Keyboard trap activated");
#endif
}

/*
**  Called to deactivate trapping the system keys, but leave ISR installed.
**  If the user had attempted a PrintScreen operation while the trap was
**  active, it is now called.
*/

void deactivate_KBtrap(void)
{
      KBtrap_active = False_;
#ifdef TEST
      puts("Keyboard trap deactivated");
#endif
      if (PrtScrnPending)
      {
            if (Success_ == PrtScrn())
                  PrtScrnPending = False_;
      }
}

/*
**  Traps system keys - Ignores Ctrl-Alt-Del, Ctrl-C, Ctrl-Break, SysRq,
**                      and Pause.
**
**                      Defers execution of PrintScreen
*/
```

```
101 |
102 | static void INTERRUPT FAR newkbisr(void)
103 | {
104 |       static int count = 0;
105 |       int key = keyport();
106 |
107 |       if (KBtrap_active)
108 |       {
109 |             if (count)
110 |             {
111 |                   unsigned char kbin;
112 |
113 |                   --count;
114 |
115 |                   if (PRTSCRNSCAN == key)
116 |                         PrtScrnPending = True_;
117 | IGNORE:
118 |                   kbin = (unsigned char)inp(KEYPORT+1); /* reset keyboard  */
119 |                   outp(KEYPORT+1, kbin|0x80);
120 |                   outp(KEYPORT+1, kbin);
121 |                   disable();
122 |                   outp(CONTROLLERPORT,0x20); /* tell controller to shut up */
123 |                   enable();
124 |                   return;
125 |             }
126 |             else switch (key)
127 |             {
128 |             case MULTI2SCAN:
129 |                   count = 1;
130 |                   goto IGNORE;
131 |
132 |             case MULTI3SCAN:
133 |                   count = 2;
134 |                   goto IGNORE;
135 |
136 |             case DELSCAN:
137 |                   if (CTRLALT == (kbstat() & CTRLALT))
138 |                         goto IGNORE;
139 |                   break;
140 |
141 |             case PRTSCRNSCAN:
142 |                   PrtScrnPending = True_;
143 |                   goto IGNORE;
144 |
145 |             case CSCAN:
146 |             case BREAKSCAN:
147 |                   if (CTRL == (kbstat() & CTRL))
148 |                         goto IGNORE;
149 |                   break;
150 |
151 |             case SYSREQSCAN:
152 |                   goto IGNORE;
153 |             }
154 |       }
155 |       else
156 |       {
157 |             if (count)
158 |             {
159 |                   --count;
160 |                   return;
161 |             }
162 |       }
163 |       oldkbisr();                         /* chain to old keyboard isr */
164 | }
165 |
166 | static void KBtrap(void)
167 | {
168 |       install();
169 |       atexit(cleanup);
170 |       KBtrap_installed = True_;
171 | #ifdef TEST
172 |       puts("Keyboard trap installed");
173 | #endif
174 | }
175 |
176 | static void cleanup(void)
177 | {
178 |       uninstall();
179 | #ifdef TEST
180 |       puts("Keyboard trap uninstalled");
181 | #endif
182 | }
183 |
184 | #ifdef TEST
185 |
186 | #include <stdio.h>
187 | #include <conio.h>
188 | #include <signal.h>
189 | #include <errno.h>
190 |
191 | main()
192 | {
193 |       int ch = 0;
194 |
195 |       puts("This is a test of Ctrl-Alt-Del disabling.");
196 |       puts("Press any key, but only Esc should stop this program.");
197 |
198 |       activate_KBtrap();
199 |
200 |       while (0x1b != ch)
```

```
 201 |       {
 202 |             if (kbhit())
 203 |             {
 204 |                   ch = getch();
 205 |                   printf("key value = %02Xh, PrtScrnPending = %d\n",
 206 |                         ch, PrtScrnPending);
 207 |             }
 208 |       }
 209 |
 210 |       deactivate_KBtrap();
 211 |
 212 |       return EXIT_SUCCESS;
 213 | }
 214 |
 215 | #endif /* TEST */
```

TEXT STATISTICS

```
  6196 characters
   215 lines
```

LEXICAL STATISTICS

```
    31 comments [std-C]
     0 comments [C++]
    35 preprocessor instructions
     0 constants [character]
    10 constants [string]
    26 constants [numeric]
```

SYMBOL TABLE

```
    ALT             18
    BREAKSCAN       21     146
    Boolean_T       45      51      52
    CONTROLLERPORT           27     122
    CSCAN           20     145
    CTRL            17     147+
    CTRLALT         16     137+
    DELSCAN         19     136
    EXIT_SUCCESS            212
    FAR             42      56     102
    False_          45      52      84      91
    IGNORE         117     130     134     138     143     148     152
    INTERRUPT       42      56     102
    KBtrap          54      69     166
    KBtrap_active            51      70      84     107
    KBtrap_installed         52      68     170
    KEYPORT         26      30     118     119     120
    MULTI2SCAN               24     128
    MULTI3SCAN               25     132
    PRTSCRNSCAN              22     115     141
    Peekw           28
    PrtScrn         90
    PrtScrnPending           45      88      91     116     142     206
    SYSREQSCAN               23     151
    Success_        90
    TEST            71      85     171     179     184
    True_           70     116     142     170
    activate_KBtrap          66     198
    atexit         169
    ch             193     200     204     206
    cleanup         48      55     169     176
    conio          187
    count          104     109     113     129     133     157     159
    deactivate_KBtrap        82     210
    disable        121
    dos             11
    enable         123
    errno          189
    getch          204
    getvect         34
    h       10      11     186     187     188     189
    inp             30     118
    install         34     168
    kbhit          202
    kbin           111     118     120
    kbin|0x80      119
    kbstat          28     137     147
    key            105     115     126
    keyport         30     105
    main           191
    newkbisr        34      56     102
    oldkbisr        34      38      42     163
    outp           119     120     122
    printf         205
    puts            72      86     172     180     195     196
    setvect         34      38
    signal         188
    stdio          186
    stdlib          10
    uninstall       38     178
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  KBTRAP.H - Header for KBTRAP.C
   5 | */
   6 |
   7 | #ifndef KBTRAP_H__
   8 | #define KBTRAP_H__
   9 |
  10 | void activate_KBtrap(void);
  11 | void deactivate_KBtrap(void);
  12 |
  13 |
  14 | #endif /* KBTRAP_H__ */
```

TEXT STATISTICS

```
   221 characters
    14 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    KBTRAP_H__               7       8
    activate_KBtrap         10
    deactivate_KBtrap       11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  by: Dan Kozak
**  Revisions:
**  30-Mar-96  Ed Blackman  OS/2 mods
*/

/*
**  For use with your code, strip out the demo main() and make this into
**  a header file.
*/

#ifndef KB_DATA__H
#define KB_DATA__H

#include "extkword.h"
#include "snipkbio.h"

typedef struct                          /* Keyboard status structure  */
{
  unsigned short                        /* Least Significant Bit      */
  right_shift_down  : 1,                /* Right Shift key depressed  */
  left_shift_down   : 1,                /* Left Shift key depressed   */
  ctrl_down         : 1,                /* Ctrl key depressed         */
  alt_down          : 1,                /* Alt key depressed          */
  scroll_on         : 1,                /* Scroll Lock is on          */
  num_on            : 1,                /* Num Lock is on             */
  caps_on           : 1,                /* Caps Lock is on            */
  ins_on            : 1,                /* Insert state is active     */
  left_ctl          : 1,                /* Left Ctl key depressed     */
  left_alt          : 1,                /* Left Alt key depressed     */
#if defined (__OS2__)
  right_ctl         : 1,                /* Right Ctl key depressed    */
  right_alt         : 1,                /* Right Alt key depressed    */
#else       /* assume DOS */
  sys_rq            : 1,                /* SysRq depressed            */
  pause_on          : 1,                /* Pause is active            */
#endif
  scroll_down       : 1,                /* Scroll Lock key depressed  */
  num_down          : 1,                /* Num Lock key depressed     */
  caps_down         : 1,                /* Caps Lock key depressed    */
#if defined (__OS2__)
  sys_rq            : 1,                /* SysRq depressed            */
#else       /* assume DOS */
  ins_down          : 1;                /* Insert key depressed       */
#endif
} biosshiftstate;                       /* Most Significant Bit       */

biosshiftstate FAR * volatile kbd_status =
      (biosshiftstate FAR * volatile)(peekkey());

#ifdef TEST

#include <stdio.h>
#include <time.h>

main()
{
      clock_t start = clock();

      puts("Press some key stuff and I'll tell you what in 3 seconds...\n");

      while (((clock() - start) / CLOCKS_PER_SEC) < 3)
#if defined(__OS2__)
          /* to store key info where peekkey() can find it */
            if(kbhit()) ext_getch()
#endif
            ;


      printf("right_shift_down = %d\n",kbd_status->right_shift_down);
      printf("left_shift_down = %d\n",kbd_status->left_shift_down);
      printf("ctrl_down = %d\n",kbd_status->ctrl_down);
      printf("alt_down = %d\n",kbd_status->alt_down);
      printf("scroll_on = %d\n",kbd_status->scroll_on);
      printf("num_on = %d\n",kbd_status->num_on);
      printf("caps_on = %d\n",kbd_status->caps_on);
      printf("ins_on = %d\n",kbd_status->ins_on);
      printf("filler = %d\n",kbd_status->filler);
      printf("ctrl_numloc = %d\n",kbd_status->ctrl_numloc);
      printf("scroll_down = %d\n",kbd_status->scroll_down);
      printf("num_down = %d\n",kbd_status->num_down);
      printf("caps_down = %d\n",kbd_status->caps_down);
      printf("ins_down = %d\n",kbd_status->ins_down);

      return 0;
}

#endif /* TEST */

#endif /* KB_DATA__H */
```

TEXT STATISTICS

3479 characters
92 lines

LEXICAL STATISTICS

30 comments [std-C]
0 comments [C++]
17 preprocessor instructions
0 constants [character]
17 constants [string]
21 constants [numeric]

SYMBOL TABLE

```
CLOCKS_PER_SEC          64
FAR             50      51
KB_DATA__H              14      15
TEST            53
__OS2__         33      43      65
alt_down        26      75
biosshiftstate          48      50      51
caps_down       42      84
caps_on         29      78
clock           60      64
clock_t         60
ctrl_down       25      74
ctrl_numloc             81
defined         33      43      65
ext_getch       67
filler          80
h       55      56
ins_down        46      85
ins_on          30      79
kbd_status              50      72      73      74      75      76      77      78      79      80
        81      82      83      84      85
kbhit           67
left_alt        32
left_ctl        31
left_shift_down         24      73
main            58
num_down        41      83
num_on          28      77
pause_on        38
peekkey         51
printf          72      73      74      75      76      77      78      79      80      81      82
        83      84      85
puts            62
right_alt       35
right_ctl       34
right_shift_down        23      72
scroll_down             40      82
scroll_on       27      76
start           60      64
stdio           55
sys_rq          37      44
time            56
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  KB_STUFF.C - Functions to stuff characters and/or strings into a PC's
**               (BIOS) keyboard buffer.
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <dos.h>
#include "sniptype.h"
#include "pchwio.h"
#include "snipkbio.h"

static unsigned head, tail, start, end;
static int idx = 0;
static unsigned short keystack[16][2];

/*
**  ungetkey()
**
**  Stuffs characters into the keyboard buffer.
**
**  Parameters: 1 - Extended character to stuff
**
**  Returns: Success_ or EOF
**
**  Note: This function assumes that the keyboard buffer is in
**        the normal (for IBM) location of 40:1E.
**
*/

int ungetkey(unsigned short key)
{
      int count;

      head  = Peekw(0x40, 0x1a);
      tail  = Peekw(0x40, 0x1c);
      start = Peekw(0x40, 0x80);
      end   = Peekw(0x40, 0x82);

      count = tail - head;
      if (0 > count)
            count += (16 * sizeof(unsigned));
      count >>= 1;

      if (15 > count)
      {
            disable();
            keystack[idx][0] = Peekw(0x40, tail);
            keystack[idx][1] = tail;
            Pokew(0x40, tail, key);
            tail += sizeof(unsigned);
            if (end <= tail)
                  tail = start;
            Pokew(0x40, 0x1c, tail);
            enable();
            return key;
      }
      return EOF;
}

/*
**  KB_stuff()
**
**  Stuffs strings into the keyboard buffer.
**
**  Parameters: 1 - String to stuff
**
**  Returns: Success_ if successful
**           Error_   in case of error, plus keyboard buffer is
**                    restored
**
**  Note: This function assumes that the keyboard buffer is in
**        the normal (for IBM) location of 40:1E.
*/

int KB_stuff(char *str)
{
      int ercode = Success_;

      idx = 0;
      while (*str)
      {
            if (EOF == ungetkey((unsigned)(*str++)))
            {
                  disable();
                  while (0 <= --idx)
                  {
                        tail = keystack[idx][1];
                        Pokew(0x40, tail, keystack[idx][0]);
                  }
                  Pokew(0x40, 0x1c, tail);
                  enable();
                  ercode = Error_;
```

```
 101 |                   break;
 102 |              }
 103 |              else  ++idx;
 104 |       }
 105 |       idx = 0;
 106 |       return ercode;
 107 | }
```

TEXT STATISTICS

```
  2671 characters
   107 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
    29 constants [numeric]
```

SYMBOL TABLE

```
    EOF             65      90
    Error_         100
    KB_stuff        83
    Peekw           42      43      44      45      55
    Pokew           57      61      96      98
    Success_        85
    count           40      47      48      49      50      52
    disable         54      92
    dos             15
    enable          62      99
    end             20      45      59
    ercode          85     100     106
    h       15
    head            20      42      47
    idx             21      55      56      87      93      95      96     103     105
    key             38      57      63
    keystack        22      55      56      95      96
    start           20      44      60
    str             83      88      90
    tail            20      43      47      55      56      57      58      59      60      61      95
            96      98
    ungetkey        38      90
```

```
/* +++Date last modified: 05-Jul-1997 */

#include "extkword.h"
#include "snipkbio.h"

static volatile unsigned char FAR *keyflags =
      (unsigned char FAR *)0x00400017L;

/*
**  Caps Lock
*/

void setcaps(void)
{
      *keyflags |= 0x40;
}

void clrcaps(void)
{
      *keyflags &= ~0x40;
}

/*
**  Num Lock
*/

void setnumlock(void)
{
      *keyflags |= 0x20;
}

void clrnumlock(void)
{
      *keyflags &= ~0x20;
}

/*
**  Scroll Lock
*/

void setscrlock(void)
{
      *keyflags |= 0x10;
}

void clrscrlock(void)
{
      *keyflags &= ~0x10;
}
```

```
TEXT STATISTICS

   595 characters
    49 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     2 constants [string]
     7 constants [numeric]

SYMBOL TABLE

    FAR              6       7
    clrcaps         18
    clrnumlock               32
    clrscrlock               46
    keyflags         6      15      20      29      34      43      48
    setcaps         13
    setnumlock               27
    setscrlock               41
```

```
/* +++Date last modified: 05-Jul-1997 */

/* keywatch.c 12-29-91 Robert Mashlan, Public Domain               *\

   DOS compiler portability modifications added by Bob Stout,
   1:106/2000.6

   This program monitors the keyboard interrupt, and stores the
   status of each key as to whether it is pressed or released.

   This is done by capturing interrupt 9, and watching the make/break
   codes.  The status is updated in the keys array, where 1 means
   that the key is pressed, while 0 means the key is released.  The
   key array is uses the scan code for an index instead of the ascii
   character.  It is simple enough to find the scan code for a key,
   just run this program and watch the display.

   The ekeys array will reflect the status of keys found on an AT
   keyboard.  For instance, the left and right alt keys are
   differentiated, as well as the edit control keys on the numeric
   keypad and the one not on the numeric keypad.

   Since this program installs an interrupt handler, it should be
   terminated normally, such the keyboard handler can be removed.
   The ^C/^Break exit is captured via signal(), but all possible
   exits should be trapped.

\*                                                                 */

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <signal.h>
#include "sniptype.h"
#include "pchwio.h"

volatile char keys[128];           /* array of key states          */
volatile char ekeys[128];          /* array of AT key states       */

#define KEYPORT        0x60        /* keyboard scan code port      */
#define keyport()      inp(KEYPORT)
        /* macro that returns the scancode of the key that caused  */
        /* the interrupt                                           */

/* Define:                                                        *\

   installisr()
         installation macro, installs newkbisr() in the keyboard
         interrupt chain

   removeisr()
         removal macro, call to remove newkbisr() from interrupt
         chain.  oldkbisr()  must be removed before program ends
\*                                                                 */

#ifdef __ZTC__
 #define installisr() int_intercept(0x09, newkbisr, 0)
 #define removeisr()  int_restore(0x09);
#else
 #define installisr()  (oldkbisr=getvect(0x09),setvect(0x09,newkbisr))
 #define removeisr()   setvect(0x09,oldkbisr)
 #ifdef __TURBOC__
  void INTERRUPT (FAR *oldkbisr)(void);    /* address of old ISR   */
 #else
  void (INTERRUPT FAR *oldkbisr)(void);    /* address of old ISR   */
 #endif
#endif

#if defined(__ZTC__)
 int newkbisr(struct INT_DATA *pd)
#elif defined(__TURBOC__)
 void INTERRUPT newkbisr(void)
#else
 void INTERRUPT FAR newkbisr(void)
#endif
{
      static extkey;
      BYTE scancode = (BYTE)keyport();         /* read keyboard scan code */

      if (scancode == 0xe0)
            extkey = 1;              /* use ekey array on next scan code */
      else
      {
            if (scancode & 0x80)                          /* key released */
                  (extkey ? ekeys : keys)[scancode & 0x7f] = 0;
            else  (extkey ? ekeys : keys)[scancode] = 1;
            extkey = 0;
      }

#ifdef __ZTC__
      return 0;                        /* chain to previous keyboard ISR */
#else
      oldkbisr();                      /* chain to previous keyboard ISR */
#endif
}

int keyspressed(void)          /* returns number of keys being held down */
{
      int i, result = 0;
```

```
101 |       for (i = 0; i < 128; i++)
102 |       {
103 |             result += keys[i];
104 |             result += ekeys[i];
105 |       }
106 |       return result;
107 | }
108 |
109 | int main(void)
110 | {
111 |       int lastkeycount = 0;
112 |
113 |       signal(SIGINT,SIG_IGN);  /* ignore ^C and ^Break */
114 |       installisr();            /* install interrupt handler */
115 |       while(1)
116 |       {
117 |             int i;
118 |
119 |             if (keyspressed() != lastkeycount) /* change in keystatus */
120 |             {
121 |                   lastkeycount = keyspressed();
122 |                   puts("---");
123 |                   for(i = 0; i < 128; i++)
124 |                   {
125 |                         if (keys[i])
126 |                               printf("key with scan code %02x "
127 |                                     "has been pressed\n", i);
128 |                         if (ekeys[i])
129 |                               printf("key with scan codes e0 %02x "
130 |                                     "had been pressed\n", i);
131 |                   }
132 |             }
133 |             if (kbhit() && getch()==0x1b) /* terminate when Esc pressed */
134 |                   break;
135 |       }
136 |       removeisr();   /* remove interrupt handler */
137 |       return 0;
138 | }
```

TEXT STATISTICS

4737 characters
138 lines

LEXICAL STATISTICS

21 comments [std-C]
0 comments [C++]
25 preprocessor instructions
0 constants [character]
7 constants [string]
26 constants [numeric]

SYMBOL TABLE

BYTE		78+								
FAR		63	65	74						
INTERRUPT		63	65	72	74					
INT_DATA		70								
KEYPORT		40	41							
SIGINT		113								
SIG_IGN		113								
__TURBOC__		62	71							
__ZTC__		56	69	90						
conio		32								
defined		69	71							
dos		31								
ekeys		38	85	86	104	128				
extkey		77	81	85	86	87				
getch		133								
getvect		60								
h	30	31	32	33						
i	99	101+	103	104	117	123+	125	127	128	130
inp		41								
installisr		57	60	114						
int_intercept		57								
int_restore		58								
kbhit		133								
keyport		41	78							
keys		37	85	86	103	125				
keyspressed		97	119	121						
lastkeycount		111	119	121						
main		109								
newkbisr		57	60	70	72	74				
oldkbisr		60	61	63	65	93				
pd		70								
printf		126	129							
puts		122								
removeisr		58	61	136						
result		99	103	104	106					
scancode		78	80	84	85	86				
setvect		60	61							
signal		33	113							
stdio		30								

```
/* +++Date last modified: 05-Jul-1997 */

/*
** KILLFF.C - A program was written to strip out all the Form Feeds
** in text files.
**
** Public domain by Erik VanRiper, 12/22/91
** Modified by Bob Stout, 17 Feb 93
**
** Reads a text file and makes a duplicate with NO Form Feed
** characters! The default action is to create a duplicate without
** Form Feeds, then remove the original and rename the duplicate,
** although an explicit output file name may be specified.
**
** Form Feed characters are replaced with newline characters ('\n').
** Since ANSI mandates that fwrite() will translate newlines when
** a stream is opened in text (non-binary) mode, these will appear
** in the output file in a format appropriate to the implementation,
** e.g. CRLF pairs on PC's.
**
** Usage: KILLFF filename [newname]
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define BSIZ 32768U             /* max size of read/write buffer */

main(int argc, char *argv[])
{
   FILE *in, *out;              /* input and output files        */
   char name[80],               /* name of file to be fixed      */
        temp[80],               /* output file name              */
        *buf;                   /* buffer we will use to write   */
   size_t bad,                  /* check to see if write ok      */
          num;                  /* number of bytes read          */
   int retval  = EXIT_SUCCESS,  /* return value                  */
       tmpflag = 0;             /* non-zero if tmpnam() used     */

   printf("\nKILL FORM FEEDS by Erik VanRiper & Bob Stout\n\n");

   if(argc < 2)                                /* usage info       */
   {
      puts("Usage: KILLFF input_file [output_file]");
      puts("\nIf no output file is given, the input file will "
           "be replaced.");
      return retval;                           /* return to OS     */
   }

   strcpy(name,argv[1]);                       /* input filename   */
   if(argc == 3) strcpy(temp,argv[2]);         /* outfile name     */
   else
   {
      tmpnam(temp);
      tmpflag = -1;
   }

   if((in = fopen(name,"r")) == NULL)          /* Open in file     */
   {
      printf("\nCan't Open Input File %s",name);
      return (retval = EXIT_FAILURE);          /* return to OS     */
   }
   if((out = fopen(temp,"wt")) == NULL)        /* open out file    */
   {
      printf("\nCan't Open Output File %s",temp);
      fclose(in);                              /* close in file    */
      return (retval = EXIT_FAILURE);          /* return to OS     */
   }

   if((buf = malloc(BSIZ)) == NULL)     /* malloc a large buffer */
   {
      printf("\nOut of memory\n");
      return (retval = EXIT_FAILURE);          /* return to OS     */
   }

   printf("Input file: %s Output file: %s\n",
      name,tmpflag ? name : temp);

   /* read in file while chars to read */

   while (0 < (num = fread(buf,sizeof(char),BSIZ,in)))
   {
      size_t i;

      for (i = 0; i < num; ++i)                /* look for FF      */
         if ('\f' == buf[i])
            buf[i] = '\n';                    /* change to newline */

      bad=fwrite(buf,sizeof(char),num,out);  /* write out buf    */
      if(bad != num)                           /* error            */
      {
         printf("\nCan't Write to %s ", temp);
         retval = EXIT_FAILURE;                /* return to OS     */
         break;
      }
   }
   fclose(in);                                 /* close in file    */
   fclose(out);                                /* close out file   */
   free(buf);                                  /* free memory      */
```

```
 101 |    if (tmpflag)
 102 |    {
 103 |       if (remove(name))
 104 |       {
 105 |          printf("Can't rename %s\n", name);
 106 |          printf("Converted file is named %s\n", temp);
 107 |       }
 108 |       else
 109 |          rename(temp, name);
 110 |    }
 111 |    printf("\nDone!");                        /* Finished         */
 112 |    return retval;                            /* return to OS     */
 113 | }
 114 | /*
 115 |
 116 | List this source file to test this program!
 117 |
 118 | New page
 119 |
 120 | New page
 121 |
 122 | All done
 123 |
 124 | */
```

TEXT STATISTICS

```
  4128 characters
   124 lines
```

LEXICAL STATISTICS

```
    34 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     2 constants [character]
    14 constants [string]
    11 constants [numeric]
```

WARNINGS

```
     line   116 : control character ignored
     line   118 : control character ignored
     line   120 : control character ignored
```

SYMBOL TABLE

```
    BSIZ            28      71      82
    EXIT_FAILURE            62      68      74      94
    EXIT_SUCCESS            38
    FILE            32
    NULL            59      64      71
    argc            30      43      52
    argv            30      51      52
    bad             36      90      91
    buf             35      71      82      87      88      90     100
    fclose          67      98      99
    fopen           59      64
    fread           82
    free           100
    fwrite          90
    h       24      25      26
    i       84      86+     87      88
    in              32      59      67      82      98
    main            30
    malloc          71
    name            33      51      59      61      78+    103     105     109
    num             37      82      86      90      91
    out             32      64      90      99
    printf          41      61      66      73      77      93     105     106     111
    puts            45      46
    remove         103
    rename         109
    retval          38      48      62      68      74      94     112
    size_t          36      84
    stdio           24
    stdlib          26
    strcpy          51      52
    string          25
    temp            34      52      55      64      66      78      93     106     109
    tmpflag         39      56      78     101
    tmpnam          55
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** public domain demo of cooperative multitasking using function pointers
** written 29 Nov, 1992
** by David Gersic
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#define MAX_TASKS 32               /* maximum number of runnable tasks */

/* global variables */

void (*tasks[MAX_TASKS])(void);    /* pointer to list of tasks to run  */

/* prototypes */

int  schedule(void (*fp)(void));   /* add function to the task list    */
int  mark(void (*fp)(void));       /* mark task in list                */
int  kill(void (*fp)(void));       /* remove function from task list   */
void dispatcher(void);             /* dispatch a task                  */
int  set_up(void);                 /* initialization code              */
int  terminate(void);              /* shutdown code                    */

int main(void)
{
      void walk_the_dog(void);
      void floss_the_cat(void);
      void make_the_donuts(void);

      if(set_up())
      {
            fputs("Initialization failed. Exiting.\n",stderr);
            exit(1);
      }

      if(schedule(walk_the_dog))
            fputs("Could not add new function to task list.\n",stderr);
      if(schedule(floss_the_cat))
            fputs("Could not add new function to task list.\n",stderr);
      if(schedule(make_the_donuts))
            fputs("Could not add new function to task list.\n",stderr);

      while(!kbhit())  /* run 'till key hit */
            dispatcher();

      if(terminate())
      {
            fputs("Error while shutting down.\n",stderr);
            exit(1);
      }
      return(0);
}

/*
** initialize task buffer and any other 'internal' setup code
*/

int set_up(void)
{
      memset(tasks, 0x00, sizeof(tasks));
      return(0);
}

/*
** kill any tasks still in the table and shut down
*/

int terminate(void)
{
      int i;

      for(i = 0; i < MAX_TASKS; i++)
      {
            if (tasks[i])
                  kill(tasks[i]);
      }
      return(0);
}

/*
** dispatch the next runnable task
*/

void dispatcher(void)
{
      static int i = 0;

      if(tasks[i])
            tasks[i]();
      i = (i + 1) % MAX_TASKS;
}

/*
** add task to the list of runnable tasks
*/
```

```
101 |
102 | int schedule(void (*fp)(void))
103 | {
104 |       int i;
105 |
106 |       for (i = 0; i < MAX_TASKS && tasks[i]; i++)
107 |             ;                         /* find a open slot in the table    */
108 |       if (i < MAX_TASKS)              /* if not end of table              */
109 |             tasks[i] = fp;            /* add task to list                 */
110 |       else  return(1);
111 |       return(0);
112 | }
113 |
114 | /*
115 | ** remove a task from the list of runnable tasks
116 | */
117 |
118 | int kill(void (*fp)(void))
119 | {
120 |       int i;
121 |
122 |       for (i = 0; i < MAX_TASKS; i++)
123 |       {
124 |             if (tasks[i] == fp)
125 |             {
126 |                   tasks[i] = NULL;
127 |                   return(0);
128 |             }
129 |       }
130 |       return(1);
131 | }
132 |
133 | void walk_the_dog(void)
134 | {
135 |       puts("dog");
136 |       return;
137 | }
138 |
139 | void floss_the_cat(void)
140 | {
141 |       puts("cat");
142 |       return;
143 | }
144 |
145 | void make_the_donuts(void)
146 | {
147 |       puts("donuts");
148 |       return;
149 | }
```

```
TEXT STATISTICS

  3253 characters
   149 lines

LEXICAL STATISTICS

    21 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     8 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    MAX_TASKS       14      18      77      95     106     108     122
    NULL           126
    conio           12
    dispatcher              25      49      89
    exit            38      54
    floss_the_cat           32      43     139
    fp              22      23      24     102     109     118     124
    fputs           37      42      44      46      53
    h        9      10      11      12
    i       75      77+     79      80      91      93      94      95+    104     106+    108     109
           120     122+    124     126
    kbhit           48
    kill            24      80     118
    main            29
    make_the_donuts         33      45     145
    mark            23
    memset          65
    puts           135     141     147
    schedule        22      41      43      45     102
    set_up          26      35      63
    stderr          37      42      44      46      53
    stdio            9
    stdlib          10
    string          11
    tasks           18      65+     79      80      93      94     106     109     124     126
    terminate       27      51      73
    walk_the_dog            31      41     133
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**        L2ROMAN.C  -  Converts long integers to Roman numerals
**
**        Jim Walsh, Dann Corbit, Bob Stout, and Others made this.
**
**        This Program Is Released To The Public Domain
**
** -------------------------------------------------------------------
** Compiling:
**    If the symbol ALLOW_BAR_NOTATION is defined, then bar notation is
**    allowed.  If the Romans wanted to record one million, they did not
**    write down 1000 M's.  They would right one M with a bar over it.
**    Any Roman numeral with a bar over it is multiplied by 1000.
**    Unfortunately, this is not a standard character in most character sets.
**    If you choose to #define ALLOW_BAR_NOTATION, then you will have to
**    translate the symbols to final form yourself.
*/

#include <stdlib.h>
#include "sniptype.h"

typedef struct tag_RomanToDecimal {
      long  PostValue;
      char *PostLetter;
      long  PreValue;
      char *PreLetter;
} R2D;

/*
**  Set PrefixesAreOK = True_ to enable prefix notation (e.g. 4 = "IV")
**  Set PrefixesAreOK = False_ to disable prefix notation (e.g. 4 = "IIII")
*/

static Boolean_T PrefixesAreOK = True_;

static R2D RomanConvert[] = {
#if defined(ALLOW_BAR_NOTATION)
        /*
        ** A Roman numeral with a bar over it is
        ** that value multiplied by 1000.
        */
        {1000000L, "<M-bar>", 900000L, "<CM-bar>"},
        {500000L, "<D-bar>", 400000L, "<CD-bar>"},
        {100000L, "<C-bar>", 90000L, "<XC-bar>"},
        {50000L, "<L-bar>", 40000L, "<XL-bar>"},
        {10000L, "<X-bar>", 9000L, "<IX-bar>"},
        {5000L, "<V-bar>", 4000L, "<IV-bar>"},
#endif
        {1000L, "M", 900L, "CM"},
        {500L, "D", 400L, "CD"},
        {100L, "C", 90L, "XC"},
        {50L, "L", 40L, "XL"},
        {10L, "X", 9L, "IX"},
        {5L, "V", 4L, "IV"},
        {1L, "I", 1L, "I"}
    };

/*
**  long2roman() - Convert a long integer into roman numerals
**
**  Arguments: 1 - Value to convert
**             2 - Buffer to receive the converted roman numeral string
**             3 - Length of the string buffer
**
**  Returns: Pointer to the buffer, else NULL if error or buffer overflow
*/

char *long2roman(long val, char *buf, size_t buflen)
{
      size_t posn = 0;
      size_t place = 0;

#if !defined(ALLOW_BAR_NOTATION)
      if (val > 3999L)
            return NULL;
#endif
      do
      {
            while (val >= RomanConvert[place].PostValue)
            {
                  posn += sprintf(&buf[posn], "%s",
                                  RomanConvert[place].PostLetter);
                  val -= RomanConvert[place].PostValue;
                  if (posn >= buflen)
                        return NULL;
            }
            if (PrefixesAreOK)
            {
                  if (val >= RomanConvert[place].PreValue)
                  {
                        posn += sprintf(&buf[posn], "%s",
                                        RomanConvert[place].PreLetter);
                        val -= RomanConvert[place].PreValue;
                        if (posn >= buflen)
                              return NULL;
                  }
            }
            place++;
```

```
 101 |       } while (val > 0);
 102 |
 103 |       return buf;
 104 | }
 105 |
 106 | #ifdef TEST
 107 |
 108 | #include <stdio.h>
 109 |
 110 | int main(int argc, char *argv[])
 111 | {
 112 |       long value;
 113 |       char buf[128];
 114 |
 115 |       while (--argc)
 116 |       {
 117 |             value = atol(*(++argv));
 118 |             printf("\n%ld = %s\n", value, long2roman(value, buf, 128));
 119 |       }
 120 |
 121 |       return 0;
 122 | }
 123 |
 124 | #endif /* TEST */
```

TEXT STATISTICS

```
  3664 characters
   124 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
    30 constants [string]
    33 constants [numeric]
```

SYMBOL TABLE

```
    ALLOW_BAR_NOTATION              39      75
    Boolean_T       36
    NULL            77      87      97
    PostLetter              26      84
    PostValue       25      81      85
    PreLetter       28      94
    PreValue        27      91      95
    PrefixesAreOK           36      89
    R2D             29      38
    RomanConvert            38      81      84      85      91      94      95
    TEST           106
    True_           36
    argc           110     115
    argv           110     117
    atol           117
    buf             70      83      93     103     113     118
    buflen          70      86      96
    defined         39      75
    h       21     108
    long2roman              70     118
    main           110
    place           73      81      84      85      91      94      95     100
    posn            72      83+     86      93+     96
    printf         118
    size_t          70      72      73
    sprintf         83      93
    stdio          108
    stdlib          21
    tag_RomanToDecimal              24
    val             70      76      81      85      91      95     101
    value          112     117     118+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ldfloor() -- long double floor
  4 | ** public domain by Raymond Gardner  Englewood, CO
  5 | ** tested with TC++
  6 | ** assumptions: 80-bit IEEE format numbers, stored LSB first
  7 | **   (Intel style), longs & ints are accessed from arbitrary boundaries
  8 | */
  9 |
 10 | #include "snipmath.h"
 11 |
 12 | long double ldfloor(long double a)
 13 | {
 14 |     long double a0;
 15 |     int e, n;
 16 |
 17 |     a0 = a;
 18 |     e = ((int *)&a)[4] & 0x7FFF;        /* extract exponent         */
 19 |     if ( e == 0 )                       /* 0 is special case        */
 20 |         return (long double) 0.0;
 21 |     e -= 16383;                         /* unbias exponent          */
 22 |     if (e < 0)                          /* if < 0, num is < 1,...   */
 23 |     {
 24 |         a = 0.0;                        /* so floor is zero         */
 25 |     }
 26 |     else if ((n = 63 - e) > 0)          /* clear n least sig. bits  */
 27 |     {
 28 |         if (n < 32)                     /* clear n lowest bits      */
 29 |         {
 30 |             ((unsigned long *)&a)[0] &= ~((1L << n) - 1);
 31 |         }
 32 |         else                            /* n >= 32 */
 33 |         {
 34 |             ((unsigned long *)&a)[0] = 0; /* clear lower 32 bits           */
 35 |             n -= 32;                    /* how many left to clear ?         */
 36 |             if (n)                      /* if any, clear n next lowest bits */
 37 |             {
 38 |                 ((unsigned long *)&a)[1] &= ~((1L << n) - 1);
 39 |             }
 40 |         }
 41 |     }
 42 |     if (a0 < 0 && a0 != a)          /* if neg. and it had fractional bits */
 43 |         a -= 1.0;                   /* adjust the floor                   */
 44 |     return a;                       /* return it                          */
 45 | }
```

TEXT STATISTICS

```
  1707 characters
    45 lines
```

LEXICAL STATISTICS

```
    16 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
    21 constants [numeric]
```

SYMBOL TABLE

```
    a       12      17      18      24      30      34      38      42      43      44
    a0              14      17      42+
    e       15      18      19      21      22      26
    ldfloor         12
    n       15      26      28      30      35      36      38
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  LINE.C
  5 | **  Simple filter to add line-numbers to files
  6 | **
  7 | **  public domain demo by Bob Stout
  8 | **
  9 | **  Syntax:
 10 | **  LINE may be used at the end of a command-line or in the middle, e.g.
 11 | **
 12 | **  <type|cat> myfile.c | line | more
 13 | **  add line-numbers and show myfile.c one screen at a time
 14 | **
 15 | **  or
 16 | **
 17 | **  <type|cat> myfile.c | line > lpt1
 18 | **  print listing with line numbers
 19 | **
 20 | **  or simply
 21 | **
 22 | **  line < myfile.c > file.out
 23 | */
 24 |
 25 | #include <stdio.h>
 26 | #include <stdlib.h>
 27 |
 28 | int main()
 29 | {
 30 |       static unsigned long linenumber = 0;
 31 |       int ch, newline = 1;
 32 |
 33 |       while (EOF != (ch = getchar()))
 34 |       {
 35 |             if (newline)
 36 |             {
 37 |                   printf("%06lu: ", ++linenumber);
 38 |                   newline = 0;
 39 |             }
 40 |             putchar(ch);
 41 |             if ('\n' == ch)
 42 |                   newline = 1;
 43 |       }
 44 |       return EXIT_SUCCESS;
 45 | }
```

TEXT STATISTICS

```
   945 characters
    45 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     1 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    EOF             33
    EXIT_SUCCESS              44
    ch              31      33      40      41
    getchar         33
    h       25      26
    linenumber              30      37
    main            28
    newline         31      35      38      42
    printf          37
    putchar         40
    stdio           25
    stdlib          26
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLD.c           Generic Doubly Linked Lists for fixed size data.
                    Each List has its own specific data size.
                    This version uses dummy head and dummy tail nodes.
                    Which prevents special handling for the first and last
                    nodes.

                    v1.00  94-08-21
                    v1.01  95-10-21  Changed ListCountInit to unsigned.

                    Compile with NDEBUG not defined for debugging version.
                    Compile with NDEBUG defined for production version.

                    The node pointers are restricted to valid values.
                    They point only in empty lists to invalid data.

                    Possible future enhancements:
                    - List(s) of free nodes for fast node memory alloc.
                    - FindFirst() & FindNext().
                    - Data access via first and/or last node pointers.
                      (duplicate the functions and change .current to
                      .first or .last)
                    - Node deletion via first and/or last node pointers.
                      (as for access functions, then simplify ...)

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  |  \  ------------------------------------------------------------ */

#include <stdarg.h>         /* variable arg handling */
#include <assert.h>         /* debugging */

#if defined( __TURBOC__ ) && !defined( __BORLANDC__ )
 #include <stdio.h> /* required for early Turbo compilers with assert() */
#endif

#include "ll_defs.h"        /* debugging incl MALLOC (re-) definition   */
#include "LLD.h"            /* also includes extkword.h if necessary    */

#define NO_PROBLEMS LIST_NO_PROBLEMS /* local redefinition */

struct Node
{
    struct Node * next;
    struct Node * prev;
    int data;               /* also place-holder for larger items.      */
};                          /* actual nodes have various sizes,         */
                            /* but a fixed size within a list.          */
struct ListHead
{
    struct Node * current;  /* points to the actual current node        */
    struct Node * first;    /* always points to dummy head node         */
    struct Node * last;     /* always points to dummy tail node         */
    int itemsize ;          /* zero value: used as 'list not used' flag */
};

#define ERR_MEMORY          -1

#define NODE_MALLOC(list)   (struct Node *) \
                            MALLOC( ListControl[ list ].itemsize \
                                    + 2 * sizeof( struct Node * ), char )

#define NODE_FREE(node)     FREE(node)

/* ---- Local data ---------------------------------------------------- */

static struct ListHead * ListControl = NULL;
static unsigned int ListCount ;

/* ---- LL system management ------------------------------------------ */

static int ListInit( int List, int ItemSize )
{
    struct Node * Tmp ;

    if( 0 != ItemSize )
    {
        /* create dummy head node
        */
        Tmp = NODE_MALLOC( List );
        if( NULL == Tmp )
        {
            return ERR_MEMORY ;
        }
        Tmp->prev = NULL ;     /* NULL identifies it as dummy head node */
        Tmp->data = (int)0xA709;       /* dummy value                   */
        ListControl[ List ].first = Tmp ;

        /* create dummy tail node
        */
        Tmp = NODE_MALLOC( List );
        if( NULL == Tmp )
        {
            NODE_FREE( Tmp );          /* no need to cause memory leaks */
            ListControl[ List ].first = NULL ; /* or other errors       */
            return ERR_MEMORY ;        /* even if we're in trouble ...  */
        }
        Tmp->next = NULL ;     /* NULL identifies it as dummy tail node */
```

```
101 |         Tmp->data = (int)0xA725;       /* dummy value                   */
102 |         Tmp->prev = ListControl[ List ].first ;
103 |
104 |         ListControl[ List ].current     =
105 |         ListControl[ List ].last        =
106 |         ListControl[ List ].first->next = Tmp ;
107 |     }
108 |     else
109 |     {
110 |         ListControl[ List ].current =
111 |         ListControl[ List ].first   =
112 |         ListControl[ List ].last    = NULL ;
113 |     }
114 |
115 |     ListControl[ List ].itemsize = ItemSize ; /* zero: list not used    */
116 |     return NO_PROBLEMS ;
117 | }
118 |
119 | int LLDsystemInit( unsigned ListCountInit )
120 | {
121 |     assert( (unsigned) ( ListCountInit -1 ) <= 20 -1 );
122 |                 /* higher than 20 is ridiculous for an initial setup   */
123 |                 /* zero is useless                                     */
124 |
125 |     if( NULL != ListControl )
126 |     {
127 |         return NO_PROBLEMS ; /* LL system already initialized */
128 |     }
129 |
130 |     ListControl = MALLOC( ListCountInit, struct ListHead );
131 |     if( NULL == ListControl )
132 |     {
133 |         return ERR_MEMORY ;
134 |     }
135 |
136 |     for( ListCount = 0 ; ListCount < ListCountInit ; ListCount++ )
137 |         ListInit( ListCount, 0 ); /* just mark it as usable ... */
138 |
139 |     /* ListCount is now ListCountInit */
140 |     assert( ListCount == ListCountInit );
141 |
142 |     return NO_PROBLEMS;
143 | }
144 |
145 | int LLDcreate( int ItemSize )
146 | {
147 |     unsigned List ;
148 |
149 |     assert( (unsigned) ( ItemSize -1 ) < 1024 -1 ) ;
150 |                             /* limit to 1kB. A size of 0 is ridiculous */
151 |
152 |     /* trigger automatic system initialisation if necessary
153 |     */
154 |     if( NULL == ListControl  &&  0 != LLDsystemInit( 1 ))
155 |     {
156 |         return ERR_MEMORY ;
157 |     }
158 |
159 |     /* Look for empty slot
160 |     */
161 |     for( List = 0; List < ListCount; List++ )
162 |     {
163 |         if( 0 == ListControl[ List ].itemsize )
164 |             break;
165 |     }
166 |
167 |     /*  What if NO EMPTY slot ???
168 |     */
169 |     if( List == ListCount )
170 |     {
171 |         struct ListHead * tmp ;         /* ListControl expansion needed */
172 |
173 |         tmp = MALLOC( ListCount + 1, struct ListHead );
174 |         if( NULL == tmp )
175 |         {
176 |             return ERR_MEMORY ;
177 |         }
178 |
179 |         memcpy( tmp, ListControl, ListCount * sizeof( struct ListHead ));
180 |         ListControl = tmp ;
181 |         ListCount++ ;
182 |     }
183 |
184 |     /* create dummy head node and set up ListControl for the list.
185 |     */
186 |     if( ERR_MEMORY == ListInit( List, ItemSize ))
187 |     {
188 |         return ERR_MEMORY ;
189 |     }
190 |
191 |     return (int) List ;
192 | }
193 |
194 | void LLDdelete( int List )
195 | {
196 |     struct Node * Tmp ;
197 |     struct Node * Old ;
198 |
199 |     assert( (unsigned) List < ListCount );
200 |
```

```
201 |     Tmp = ListControl[ List ].first ; /* dummies are also deleted !!!   */
202 |     while( NULL != Tmp )             /* still assuming last node has   */
203 |     {                                /* a NULL next pointer ...        */
204 |         Old = Tmp ;
205 |         Tmp = Old->next;
206 |         NODE_FREE( Old ); /* data already presumed to be deleted */
207 |     }
208 |
209 |     ListInit( List, 0 ); /* 0: mark list as not used. */
210 |
211 |     return ;
212 | }
213 |
214 | /* ---- LL system maintenance ---------------------------------------- */
215 |
216 | int LLDcheck( int List )
217 | {
218 |     if( NULL == ListControl )
219 |     {
220 |         return LIST_SYSTEM_NULL ;
221 |     }
222 |
223 |     if( (unsigned) List >= ListCount )
224 |     {
225 |         return LIST_INV_NUMBER ;
226 |     }
227 |
228 |     if( 0 == ListControl[ List ].itemsize )
229 |     {
230 |         return LIST_NOT_CREATED ;
231 |     }
232 |
233 |     if( NULL == ListControl[ List ].first
234 |         || NULL == ListControl[ List ].first->next    /* missing tail ? */
235 |         || NULL != ListControl[ List ].first->prev )
236 |     {
237 |         return LIST_ERR_HEAD ;
238 |     }
239 |
240 |     /* Validate current pointer
241 |     */
242 |     if( NULL == ListControl[ List ].current )
243 |     {
244 |         return LIST_CORRUPT7 ;    /* shouldn't be NULL with a good head */
245 |     }
246 |
247 |     if( NULL != ListControl[ List ].first->next->next ) /* empty list ? */
248 |     {                                                   /* not empty.   */
249 |         struct Node * tmp = ListControl[ List ].first ;
250 |
251 |         if( NULL == ListControl[ List ].current->next )
252 |         {
253 |             return LIST_CORRUPT6 ; /* a NULL next pointer is only valid */
254 |         }                          /* for an empty list.               */
255 |
256 |         /* look for .current in list,
257 |            checking the .prev links along the way
258 |         */
259 |         do
260 |         {
261 |             tmp = tmp->next ;
262 |
263 |             if( NULL == tmp || NULL == tmp->prev
264 |                 || tmp != tmp->prev->next )
265 |             {
266 |                 return LIST_CORRUPT5 ;  /* current not found in list */
267 |             }                           /* or link to/from next node */
268 |                                         /* invalid                   */
269 |         }while( tmp != ListControl[ List ].current );
270 |
271 |         /* Found .current in list. Also without link errors.
272 |            Now look for valid last node pointer in the list,
273 |            checking the .prev links along the way
274 |            Note that .current itself is never supposed to be equal
275 |            to .last (which points to the dummy tail) !
276 |         */
277 |         if( NULL == ListControl[ List ].last )
278 |         {
279 |             return LIST_ERR_LAST ;
280 |         }
281 |
282 |         do
283 |         {
284 |             tmp = tmp->next ;
285 |             if( NULL == tmp || NULL == tmp->prev
286 |                 || tmp != tmp->prev->next )
287 |             {
288 |                 return LIST_CORRUPT4 ;  /* last not found in list    */
289 |             }                           /* or link to/from prev node */
290 |                                         /* invalid                   */
291 |         }while( tmp != ListControl[ List ].last );
292 |
293 |         /* Found .last in list but is it really a valid last pointer?
294 |            Note: tmp == .last
295 |         */
296 |         if( NULL != tmp->next )
297 |         {
298 |             return LIST_CORRUPT3 ;
299 |         }
300 |
```

```
301 |         return NO_PROBLEMS ;
302 |     }
303 |
304 |     /* .first->next->next == NULL  =>  list is empty
305 |     */
306 |     if( ListControl[ List ].current != ListControl[ List ].first->next )
307 |     {
308 |         return LIST_CORRUPT2 ;
309 |     }
310 |
311 |     if( ListControl[ List ].last != ListControl[ List ].first->next
312 |         || ListControl[ List ].last
313 |                             != ListControl[ List ].current->prev->next )
314 |     {
315 |         return LIST_CORRUPT1 ;
316 |     }
317 |
318 |     return LIST_EMPTY ;
319 | }
320 |
321 | /* ---- node management ------------------------------------------------ */
322 |
323 | int LLDnodeInsert( int List, ... )      /* insert _BEFORE_ current node */
324 | {
325 |     va_list DataPtr ;
326 |     int Retval ;
327 |
328 |     /* set DataPtr to the address of "..."
329 |        then action, cleanup and return.
330 |     */
331 |     va_start( DataPtr, List );
332 |
333 |     Retval = LLDnodeInsertFrom( List, DataPtr );
334 |
335 |     va_end( DataPtr );
336 |     return Retval ;
337 | }
338 |
339 | int LLDnodeAdd( int List, ... )          /* insert _AFTER_ current node */
340 | {
341 |     va_list DataPtr ;
342 |     int Retval ;
343 |
344 |     /* set DataPtr to the address of "..."
345 |        then action, cleanup and return.
346 |     */
347 |     va_start( DataPtr, List );
348 |
349 |     Retval = LLDnodeAddFrom( List, DataPtr );
350 |
351 |     va_end( DataPtr );
352 |     return Retval ;
353 | }
354 |
355 | int LLDnodePrepend( int List, ... )             /* insert as first node */
356 | {
357 |     va_list DataPtr ;
358 |     int Retval ;
359 |
360 |     /* set DataPtr to the address of "..."
361 |        then action, cleanup and return.
362 |     */
363 |     va_start( DataPtr, List );
364 |
365 |     Retval = LLDnodePrependFrom( List, DataPtr );
366 |
367 |     va_end( DataPtr );
368 |     return Retval ;
369 | }
370 |
371 | int LLDnodeAppend( int List, ... )              /* insert as last node */
372 | {
373 |     va_list DataPtr ;
374 |     int Retval ;
375 |
376 |     /* set DataPtr to the address of "..."
377 |        then action, cleanup and return.
378 |     */
379 |     va_start( DataPtr, List );
380 |
381 |     Retval = LLDnodeAppendFrom( List, DataPtr );
382 |
383 |     va_end( DataPtr );
384 |     return Retval ;
385 | }
386 |
387 | int LLDnodeInsertFrom( int List, void * Source )
388 | {                                       /* insert _BEFORE_ current node */
389 |     struct Node * New ;
390 |
391 |     assert( (unsigned) List < ListCount );
392 |
393 |     /* create new node if possible
394 |     */
395 |     New = NODE_MALLOC( List );
396 |     if( NULL == New )
397 |     {
398 |         return ERR_MEMORY ;
399 |     }
400 |
```

```
401 |     /* fill node with data, link to next and previous nodes
402 |        and adjust current node pointer
403 |     */
404 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
405 |     New->next = ListControl[ List ].current;
406 |     New->prev = ListControl[ List ].current->prev;
407 |
408 |     ListControl[ List ].current->prev = New ;
409 |     New->prev->next = New ;
410 |
411 |     ListControl[ List ].current = New ;
412 |
413 |     return NO_PROBLEMS;
414 | }
415 |
416 | int LLDnodeAddFrom( int List, void * Source )
417 | {                                       /* insert _AFTER_ current node */
418 |     struct Node * New ;
419 |
420 |     assert( (unsigned) List < ListCount );
421 |
422 |     /* create new node if possible
423 |     */
424 |     New = NODE_MALLOC( List );
425 |     if( NULL == New )
426 |     {
427 |         return ERR_MEMORY ;
428 |     }
429 |
430 |     /* fill node with data and link to next and previous nodes
431 |        with special handling when the current node pointer points
432 |        to the dummy tail node: i.e it is an empty list.
433 |        (the same case in a non-empty list is made not to occur.)
434 |     */
435 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
436 |
437 |     if( NULL != ListControl[ List ].current->next )
438 |         ListControl[ List ].current = ListControl[ List ].current->next ;
439 |
440 |     New->next = ListControl[ List ].current;
441 |     New->prev = ListControl[ List ].current->prev;
442 |
443 |     ListControl[ List ].current->prev = New ;
444 |     New->prev->next = New ;
445 |
446 |     ListControl[ List ].current = New ;
447 |
448 |     return NO_PROBLEMS;
449 | }
450 |
451 | int LLDnodePrependFrom( int List, void * Source )
452 | {                                         /* insert as first node */
453 |     struct Node * New ;
454 |
455 |     assert( (unsigned) List < ListCount );
456 |
457 |     /* create new node if possible
458 |     */
459 |     New = NODE_MALLOC( List );
460 |     if( NULL == New )
461 |     {
462 |         return ERR_MEMORY ;
463 |     }
464 |
465 |     /* fill node with data and link to dummy head and actual first nodes
466 |     */
467 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
468 |     New->prev = ListControl[ List ].first;     /* == .first->next->prev */
469 |     New->next = ListControl[ List ].first->next;
470 |
471 |     ListControl[ List ].first->next = New;
472 |     New->next->prev = New ;
473 |
474 |     /* Prevent .current from pointing at the dummy tail
475 |        (New is the only normal node...)
476 |     */
477 |     if( NULL == ListControl[ List ].current->next )
478 |         ListControl[ List ].current = New;
479 |
480 |     return NO_PROBLEMS;
481 | }
482 |
483 | int LLDnodeAppendFrom( int List, void * Source )
484 | {                                          /* insert as last node */
485 |     struct Node * New ;
486 |
487 |     assert( (unsigned) List < ListCount );
488 |
489 |     /* create new node if possible
490 |     */
491 |     New = NODE_MALLOC( List );
492 |     if( NULL == New )
493 |     {
494 |         return ERR_MEMORY ;
495 |     }
496 |
497 |     /* fill node with data and link to dummy tail and actual last nodes
498 |     */
499 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
500 |     New->next = ListControl[ List ].last ;      /* == .last->prev->next */
```

```
501 |     New->prev = ListControl[ List ].last->prev;
502 |
503 |     ListControl[ List ].last->prev = New ;
504 |     New->prev->next = New ;
505 |
506 |     /* Prevent .current from pointing at the dummy tail
507 |        (New is the only normal node...)
508 |     */
509 |     if( NULL == ListControl[ List ].current->next )
510 |         ListControl[ List ].current = New;
511 |
512 |     return NO_PROBLEMS;
513 | }
514 |
515 | void LLDnodeDelete( int List )
516 | {
517 |     struct Node * Old = ListControl[ List ].current ;
518 |
519 |     assert( (unsigned) List < ListCount );
520 |
521 |     if( NULL == ListControl[ List ].current->next )
522 |     {
523 |         return ;  /* don't delete dummy tail node (list is empty) */
524 |     }
525 |
526 |     /* adjust links
527 |     */
528 |     Old->prev->next = Old->next ;
529 |     Old->next->prev = Old->prev ;
530 |
531 |     /* adjust current node pointer
532 |        prevent it from pointing to the dummy tail node
533 |     */
534 |     if( NULL != Old->next->next )
535 |         ListControl[ List ].current = Old->next ;
536 |     else
537 |         ListControl[ List ].current = Old->prev ;
538 |
539 |     NODE_FREE( Old );
540 |
541 |     return ;
542 | }
543 |
544 | int LLDnodeFind( int List, CompFunPtr Compare, void * DataPtr )
545 | {                       /* FindFirst/FindNext format may be needed ... */
546 |     int RetVal ;
547 |
548 |     assert( (unsigned) List < ListCount );
549 |
550 |     if( NULL == ListControl[ List ].first->next->next ) /* empty list ? */
551 |     {
552 |         return 2; /* a compare usually returns just -1, 0 or 1 !!! */
553 |     }
554 |
555 |     /* note: current->next will never be NULL in a non-empty list */
556 |
557 |     if( NULL == Compare ) /* default to memcmp with .itemsize */
558 |     {
559 |         while( 0 != (RetVal = memcmp( DataPtr,
560 |                                       & ListControl[ List ].current->data,
561 |                                       ListControl[ List ].itemsize ))
562 |                && NULL != ListControl[ List ].current->next->next )
563 |         {
564 |             ListControl[ List ].current=ListControl[ List ].current->next;
565 |         }
566 |         return RetVal ;
567 |     }
568 |     else
569 |     {
570 |         while( 0 != (RetVal = (*Compare)( DataPtr,
571 |                                    & ListControl[ List ].current->data ))
572 |                && NULL != ListControl[ List ].current->next->next )
573 |         {
574 |             ListControl[ List ].current=ListControl[ List ].current->next;
575 |         }
576 |         return RetVal ;
577 |     }
578 | }
579 |
580 | /* ---- current node pointer management ------------------------------ */
581 |
582 | int  LLDnodePtr2First( int List )
583 | {
584 |     assert( (unsigned) List < ListCount );
585 |
586 |     ListControl[ List ].current = ListControl[ List ].first->next ;
587 |
588 |     return NULL != ListControl[ List ].first->next->next ;
589 | }
590 |
591 | int  LLDnodePtr2Last( int List )
592 | {
593 |     assert( (unsigned) List < ListCount );
594 |
595 |     ListControl[ List ].current = ListControl[ List ].last->prev ;
596 |
597 |     return NULL != ListControl[ List ].last->prev->prev ;
598 | }
599 |
600 | int  LLDnodePtr2Next( int List )
```

```
{
    assert( (unsigned) List < ListCount );

    if( NULL == ListControl[ List ].current->next       /* empty list ? */
        || NULL == ListControl[ List ].current->next->next ) /* at end ?*/
    {
        return 0 ;              /* do not allow the current node pointer */
    }                           /* to point at the dummy tail node ...   */

    ListControl[ List ].current = ListControl[ List ].current->next ;
    return 1 ;
}

int  LLDnodePtr2Prev( int List )
{
    assert( (unsigned) List < ListCount );

    if( NULL == ListControl[ List ].current->prev       /* empty list ? */
        || NULL == ListControl[ List ].current->prev->prev ) /* begin ? */
    {
        return 0 ;              /* do not allow the current node pointer */
    }                           /* to point at the dummy head node ...   */

    ListControl[ List ].current = ListControl[ List ].current->prev ;
    return 1 ;
}

/* ---- stored data management ------------------------------------- */

int LLDnodeInt( int List )
{
    return ListControl[ List ].current->data;
}

long LLDnodeLong( int List )
{
    return *((long *) &ListControl[ List ].current->data );
}

void * LLDnodePtr( int List )
{
    return *((void **) &ListControl[ List ].current->data );
}

void FAR * LLDnodeFptr( int List )
{
    return *((void FAR **) &ListControl[ List ].current->data );
}

int LLDnodeDataTo( int List, void * Destination )
{
    if( NULL != Destination )
    {
        memcpy( Destination,
                & ListControl[ List ].current->data,
                ListControl[ List ].itemsize );
    }

    return ListControl[ List ].itemsize ;       /* size needed for blob */
}

/* ==== LLD.c  end ================================================= */
```

```
TEXT STATISTICS

 19775 characters
   662 lines

LEXICAL STATISTICS

   106 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     2 constants [string]
    27 constants [numeric]

SYMBOL TABLE

    CompFunPtr              544
    Compare         544     557     570
    DataPtr         325     331     333     335     341     347     349     351     357     363     365
           367      373     379     381     383     544     559     570
    Destination             650     652     654
    ERR_MEMORY               59      85      98     133     156     176     186     188     398     427
           462      494
    FAR             645     647
    FREE             65
    ItemSize         74      78     115     145     149     186
    LIST_CORRUPT1           315
    LIST_CORRUPT2           308
    LIST_CORRUPT3           298
    LIST_CORRUPT4           288
    LIST_CORRUPT5           266
    LIST_CORRUPT6           253
    LIST_CORRUPT7           244
    LIST_EMPTY              318
    LIST_ERR_HEAD           237
    LIST_ERR_LAST           279
    LIST_INV_NUMBER         225
    LIST_NOT_CREATED        230
    LIST_NO_PROBLEMS         42
    LIST_SYSTEM_NULL        220
    LLDcheck        216
    LLDcreate       145
    LLDdelete       194
    LLDnodeAdd              339
    LLDnodeAddFrom          349     416
    LLDnodeAppend           371
    LLDnodeAppendFrom       381     483
    LLDnodeDataTo           650
    LLDnodeDelete           515
    LLDnodeFind             544
    LLDnodeFptr             645
    LLDnodeInsert           323
    LLDnodeInsertFrom       333     387
    LLDnodeInt              630
    LLDnodeLong             635
    LLDnodePrepend          355
    LLDnodePrependFrom              365     451
    LLDnodePtr              640
    LLDnodePtr2First        582
    LLDnodePtr2Last         591
    LLDnodePtr2Next         600
    LLDnodePtr2Prev         614
    LLDsystemInit           119     154
    List             74      82      89      93      97     102     104     105     106     110     111
           112      115     147     161+    163     169     186     191     194     199     201     209
           216      223     228     233     234     235     242     247     249     251     269     277
           291      306+    311+    312     313     323     331     333     339     347     349     355
           363      365     371     379     381     387     391     395     404     405     406     408
           411      416     420     424     435     437     438+    440     441     443     446     451
           455      459     467     468     469     471     477     478     483     487     491     499
           500      501     503     509     510     515     517     519     521     535     537     544
           548      550     560     561     562     564+    571     572     574+    582     584     586+
           588      591     593     595+    597     600     602     604     605     610+    614     616
           618      619     624+    630     632     635     637     640     642     645     647     650
           655      656     659
    ListControl              62      69      89      97     102     104     105     106     110     111
           112      115     125     130     131     154     163     179     180     201     218     228
           233      234     235     242     247     249     251     269     277     291     306+    311+
           312      313     404     405     406     408     411     435     437     438+    440     441
           443      446     467     468     469     471     477     478     499     500     501     503
           509      510     517     521     535     537     550     560     561     562     564+    571
           572      574+    586+    588     595+    597     604     605     610+    618     619     624+
           632      637     642     647     655     656     659
    ListCount        70     136+    137     140     161     169     173     179     181     199     223
           391      420     455     487     519     548     584     593     602     616
    ListCountInit           119     121     130     136     140
    ListHead         51      69     130     171     173     179
    ListInit         74     137     186     209
    MALLOC           62     130     173
    NODE_FREE        65      96     206     539
    NODE_MALLOC              61      82      93     395     424     459     491
    NO_PROBLEMS              42     116     127     142     301     413     448     480     512
    NULL             69      83      87      94      97     100     112     125     131     154     174
           202      218     233     234     235     242     247     251     263+    277     285+    296
           396      425     437     460     477     492     509     521     534     550     557     562
           572      588     597     604     605     618     619     652
    New             389     395     396     404     405     406     408     409+    411     418     424
           425      435     440     441     443     444+    446     453     459     460     467     468
           469      471     472+    478     485     491     492     499     500     501     503     504+
           510
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLD.h           Generic Doubly Linked List for fixed size data-items.

                    v1.00  94-08-21
                    v1.01  95-10-21  Changed ListCountInit to unsigned.

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#ifndef LLD__H
#define LLD__H

#ifndef FAR
#include "extkword.h"
#endif

#ifndef LL__ERR_H
#define LL__ERR_H       /* same values used in LLS ...                  */

enum ListErrors         /* return values for LLDcheck()                 */
{                       /* The highest value is returned                */

    LIST_NO_PROBLEMS,   /* All is OK (multiple use)                     */
    LIST_EMPTY,         /* No data available                            */
    LIST_ERRORS,        /* Dummy to separate warnings from              */
                        /* ---- REAL errors ------------------------- */
    LIST_CORRUPT1,      /* invalid last node pointer: != first->next    */
                        /*      (empty list) or link error              */
    LIST_CORRUPT2,      /* invalid current node pointer: != first->next */
                        /*      (empty list)                            */
    LIST_CORRUPT3,      /* invalid last node pointer: Not really last.  */
    LIST_CORRUPT4,      /* invalid last node pointer: Not in list,      */
                        /*      or link error after current node        */
    LIST_ERR_LAST,      /* invalid last node pointer: NULL              */
    LIST_CORRUPT5,      /* invalid current node pointer: Not in list,   */
                        /*      or link error before current node       */
    LIST_CORRUPT6,      /* invalid current->next node pointer: NULL     */
                        /*      although the list is not empty          */
    LIST_CORRUPT7,      /* NULL current node pointer                    */
    LIST_ERR_HEAD,      /* NULL first node pointer                      */
                        /*      or error in head node                   */
    LIST_NOT_CREATED,   /* List deleted or not created                  */
    LIST_INV_NUMBER,    /* List number out of range                     */
    LIST_SYSTEM_NULL,   /* List system not initialized                  */
};

typedef int (*CompFunPtr)( const void *, const void * );
                                            /* simplifies declarations */
#endif

/* ---- LL system management and maintenance -------------------------- */
int  LLDsystemInit( unsigned ListCount );
               /* returns -1 on failure. It is not required to call it. */
               /* A second call does nothing: ListCount is ignored.     */

int  LLDcreate( int ItemSize );
                       /* returns list number to use or -1 on failure. */
                       /* MUST be called before using a list.          */
                       /* Calls LLsystemInit if necessary.             */

void LLDdelete( int List ); /* delete entire list, data is NOT free()'d */

int  LLDcheck( int List ); /* returns enum ListErrors value             */
                           /* its primary purpose is debugging.         */

/* ---- Node management ----------------------------------------------
   Functions changing current node pointer to the new node.
   Each created list has its own -- fixed -- datasize. See LLcreate().
   An ellipsis "..." indicates the data to insert.
*/
int  LLDnodeInsert( int List, ... );      /* insert BEFORE current node */
int  LLDnodeAdd( int List, ... );         /* insert AFTER current node  */
         /* a return value of -1 indicates a memory allocation problem. */

/* Functions NOT changing the current node pointer.
   Especially intended for implementation of Queue's and Stacks.
*/
int  LLDnodePrepend( int List, ... );           /* insert as first node */
int  LLDnodeAppend( int List, ... );            /* insert as last node  */
         /* a return value of -1 indicates a memory allocation problem. */

/* The following four functions are essentially the same as the preceding
   four. The data is however not passed by value but by reference.
*/
int  LLDnodeInsertFrom( int List, void * Source );
int  LLDnodeAddFrom( int List, void * Source );
int  LLDnodePrependFrom( int List, void * Source );
int  LLDnodeAppendFrom( int List, void * Source );

void LLDnodeDelete( int List );                  /* remove current node */
        /* current node ptr moved to next node. UNLESS the deleted node */
        /* was the last node: then current ptr moved to previous node   */

int  LLDnodeFind( int List, CompFunPtr Compare, void * DataPtr );
        /* Find *DataPtr in the List using the *Compare function.       */
        /* Returns the return value of *Compare. 0 == equal == found.   */
        /* non-zero == not found. Current node is set to found node.    */
```

```
101 |         /* Returns 2 for an empty list.                                */
102 |         /* First checked node is current node.                        */
103 |         /* A NULL Compare-function defaults to the use of memcmp() with */
104 |         /* the list's itemsize as third (size) parameter.             */
105 |         /* simple implementation. FindFirst and FindNext may be needed. */
106 |
107 | /* ---- current node pointer management ---------------------------------
108 |    These functions change the current node pointer and return 1 when there
109 |    was a change. A return of 0 indicates trying to move past the begin or
110 |    the end of the list, or an empty list. The return value is intended for
111 |    iteration purposes. I.e. stopping a scan through a list.
112 | */
113 | int  LLDnodePtr2First( int List );
114 | int  LLDnodePtr2Last( int List );
115 | int  LLDnodePtr2Next( int List );
116 | int  LLDnodePtr2Prev( int List );
117 |
118 | /* ---- stored data management -------------------------------------------
119 |    return typed data:
120 | */
121 | int  LLDnodeInt( int List );
122 | long LLDnodeLong( int List );
123 | void * LLDnodePtr( int List );
124 | void FAR * LLDnodeFptr( int List );
125 |
126 | /* 'return' typeless data. The return value is the size of the data.
127 |    The data is transferred to Destination.
128 |    If 'Destination' is NULL, the only action is returning the size.
129 | */
130 | int LLDnodeDataTo( int List, void * Destination );
131 |
132 | #endif /* LLD__H */
133 | /* ==== LLD.h  end ================================================== */
```

TEXT STATISTICS

```
  6610 characters
   133 lines
```

LEXICAL STATISTICS

```
    62 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CompFunPtr              50      97
    Compare         97
    DataPtr         97
    Destination            130
    FAR             16     124
    ItemSize        59
    LIST_CORRUPT1           30
    LIST_CORRUPT2           32
    LIST_CORRUPT3           34
    LIST_CORRUPT4           35
    LIST_CORRUPT5           38
    LIST_CORRUPT6           40
    LIST_CORRUPT7           42
    LIST_EMPTY              27
    LIST_ERRORS             28
    LIST_ERR_HEAD           43
    LIST_ERR_LAST           37
    LIST_INV_NUMBER         46
    LIST_NOT_CREATED        45
    LIST_NO_PROBLEMS        26
    LIST_SYSTEM_NULL        47
    LLD__H          13      14
    LLDcheck        66
    LLDcreate       59
    LLDdelete       64
    LLDnodeAdd              75
    LLDnodeAddFrom          89
    LLDnodeAppend           82
    LLDnodeAppendFrom       91
    LLDnodeDataTo          130
    LLDnodeDelete           93
    LLDnodeFind             97
    LLDnodeFptr            124
    LLDnodeInsert           74
    LLDnodeInsertFrom       88
    LLDnodeInt             121
    LLDnodeLong            122
    LLDnodePrepend          81
    LLDnodePrependFrom              90
    LLDnodePtr             123
    LLDnodePtr2First       113
    LLDnodePtr2Last        114
    LLDnodePtr2Next        115
    LLDnodePtr2Prev        116
    LLDsystemInit           55
    LL__ERR_H       20      21
    List            64      66      74      75      81      82      88      89      90      91      93
            97     113     114     115     116     121     122     123     124     130
    ListCount       55
    ListErrors              23
    Source          88      89      90      91
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ======================================================================
  4 |     LLD_BLOB.c      Generic Doubly Linked Lists for Binary Large OBjects.
  5 |                     Linked Lists for variable size data-items.
  6 |                     This is a 'front end' for the generic LLD module.
  7 |
  8 |                     v1.00  94-08-11
  9 |
 10 |  _____              This version is Public Domain.
 11 |  /_|__|             A.Reitsma, Delft, The Netherlands.
 12 |  /  |  \  ------------------------------------------------------------ */
 13 |
 14 | #include "ll_defs.h"
 15 | #include "lld.h"            /* the generic LLD functions ...            */
 16 | #include "lld_blob.h"       /* also includes extkword.h if necessary    */
 17 |
 18 | struct BlobDesc
 19 | {
 20 |     void * data ;           /* 'data' can be obtained by LLDnodePtr() ! */
 21 |     unsigned size ;
 22 | };
 23 |
 24 | #define ERR_MEMORY          -1
 25 |
 26 | /* ---- LL blob system management ------------------------------------- */
 27 |
 28 | int LLDblobCreate( void )
 29 | {
 30 |     return LLDcreate( sizeof( struct BlobDesc ));
 31 | }
 32 |
 33 | /* ---- LL blob node management -------------------------------------- */
 34 |
 35 | int LLDblobInsert( int List, void * Source, unsigned Size )
 36 | {                                   /* insert _BEFORE_ current node */
 37 |     struct BlobDesc Blob ;
 38 |
 39 |     Blob.size = Size ;
 40 |     Blob.data = malloc( Size );
 41 |
 42 |     if( NULL == Blob.data )
 43 |     {
 44 |         return ERR_MEMORY ;
 45 |     }
 46 |
 47 |     memcpy( Blob.data, Source, Size );
 48 |     LLDnodeInsertFrom( List, & Blob );
 49 |
 50 |     return LIST_NO_PROBLEMS ;
 51 | }
 52 |
 53 | int LLDblobAdd( int List, void * Source, unsigned Size )
 54 | {                                    /* insert _AFTER_ current node */
 55 |     struct BlobDesc Blob ;
 56 |
 57 |     Blob.size = Size ;
 58 |     Blob.data = malloc( Size );
 59 |
 60 |     if( NULL == Blob.data )
 61 |     {
 62 |         return ERR_MEMORY ;
 63 |     }
 64 |
 65 |     memcpy( Blob.data, Source, Size );
 66 |     LLDnodeAddFrom( List, & Blob );
 67 |
 68 |     return LIST_NO_PROBLEMS ;
 69 | }
 70 |
 71 | int LLDblobPrepend( int List, void * Source, unsigned Size )
 72 | {                                              /* insert as first node */
 73 |     struct BlobDesc Blob ;
 74 |
 75 |     Blob.size = Size ;
 76 |     Blob.data = malloc( Size );
 77 |
 78 |     if( NULL == Blob.data )
 79 |     {
 80 |         return ERR_MEMORY ;
 81 |     }
 82 |
 83 |     memcpy( Blob.data, Source, Size );
 84 |     LLDnodePrependFrom( List, & Blob );
 85 |
 86 |     return LIST_NO_PROBLEMS ;
 87 | }
 88 |
 89 | int LLDblobAppend( int List, void * Source, unsigned Size )
 90 | {                                               /* insert as last node */
 91 |     struct BlobDesc Blob ;
 92 |
 93 |     Blob.size = Size ;
 94 |     Blob.data = malloc( Size );
 95 |
 96 |     if( NULL == Blob.data )
 97 |     {
 98 |         return ERR_MEMORY ;
 99 |     }
100 |
```

```
101 |     memcpy( Blob.data, Source, Size );
102 |     LLDnodeAppendFrom( List, & Blob );
103 |
104 |     return LIST_NO_PROBLEMS ;
105 | }
106 |
107 | void LLDblobDelete( int List )
108 | {
109 |     struct BlobDesc Blob ;
110 |
111 |     LLDnodeDataTo( List, & Blob );
112 |     free( Blob.data );
113 |
114 |     LLDnodeDelete( List );
115 |
116 |     return ;
117 | }
118 |
119 | /* ---- stored data management ----------------------------------------- */
120 |
121 | unsigned LLDblobData( int List, void * Destination )
122 | {
123 |     struct BlobDesc Blob ;
124 |
125 |     LLDnodeDataTo( List, & Blob );
126 |
127 |     if( NULL != Destination )
128 |         memcpy( Destination, Blob.data, Blob.size );
129 |
130 |     return Blob.size ;       /* size needed for blob */
131 | }
132 |
133 | /* ==== LLD_BLOB.c  end ============================================== */
```

TEXT STATISTICS

```
  3404 characters
   133 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    Blob            37      39      40      42      47      48      55      57      58      60      65
           66       73      75      76      78      83      84      91      93      94      96     101
          102     109     111     112     123     125     128+    130
    BlobDesc        18      30      37      55      73      91     109     123
    Destination            121     127     128
    ERR_MEMORY              24      44      62      80      98
    LIST_NO_PROBLEMS        50      68      86     104
    LLDblobAdd              53
    LLDblobAppend           89
    LLDblobCreate           28
    LLDblobData            121
    LLDblobDelete          107
    LLDblobInsert           35
    LLDblobPrepend          71
    LLDcreate       30
    LLDnodeAddFrom          66
    LLDnodeAppendFrom      102
    LLDnodeDataTo          111     125
    LLDnodeDelete          114
    LLDnodeInsertFrom       48
    LLDnodePrependFrom              84
    List            35      48      53      66      71      84      89     102     107     111     114
           121     125
    NULL            42      60      78      96     127
    Size            35      39      40      47      53      57      58      65      71      75      76
            83      89      93      94     101
    Source          35      47      53      65      71      83      89     101
    data            20      40      42      47      58      60      65      76      78      83      94
            96     101     112     128
    free           112
    malloc          40      58      76      94
    memcpy          47      65      83     101     128
    size            21      39      57      75      93     128     130
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLD_BLOB.h      Generic Doubly Linked List for Binary Large OBjects.
                    Linked Lists for variable size data-items.

                    v1.00  94-08-11

                    - Based on the LLD module for fixed size data-items.
                    - Use the functions in the LLD module for operations
                      not specific to Blobs. You can use LLDnodePtr() to
                      obtain a pointer to the stored Blob.
                    - Note that From and To suffixes to function names are
                      implied in the Blob data related functions.

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#ifndef LLD_BLOB__H
#define LLD_BLOB__H

/* ---- LL blob system management and maintenance --------------------- */

int  LLDblobCreate( void );
                        /* returns list number to use or -1 on failure. */
                        /* MUST be called before using a list of blobs. */

/* ---- Node management ----------------------------------------------
   Functions changing current node pointer to the new node.
   A return value of -1 indicates a memory allocation problem.
*/
int  LLDblobInsert( int List, void * Source, unsigned Size );
                                          /* insert BEFORE current node */
int  LLDblobAdd( int List, void * Source, unsigned Size );
                                          /* insert AFTER current node  */

/* Functions NOT changing the current node pointer.
   Especially intended for implementation of Queue's and Stacks.
*/
int  LLDblobPrepend( int List, void * Source, unsigned Size );
                                                /* insert as first node */
int  LLDblobAppend( int List, void * Source, unsigned Size );
                                                /* insert as last node  */

void LLDblobDelete( int List );
        /* remove current node and free() the data.                    */
        /* current node ptr moved to next node. UNLESS the deleted node */
        /* was the last node: then current ptr moved to previous node   */

/* ---- stored data management ---------------------------------------
   'return' typeless data. The return value is the size of the data.
   The data is transferred to Destination.
   If 'Destination' is NULL, the only action is returning the size.
*/
unsigned LLDblobData( int List, void * Destination );

#endif /* LLD_BLOB__H */
/* ==== LLD_BLOB.h  end ================================================ */
```

```
TEXT STATISTICS

  2788 characters
    59 lines

LEXICAL STATISTICS

    17 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    Destination             56
    LLD_BLOB__H             20      21
    LLDblobAdd              35
    LLDblobAppend           43
    LLDblobCreate           25
    LLDblobData             56
    LLDblobDelete           46
    LLDblobInsert           33
    LLDblobPrepend          41
    List            33      35      41      43      46      56
    Size            33      35      41      43
    Source          33      35      41      43
```

```
/* +++Date last modified: 05-Jul-1997 */
/*
/* ======================================================================
    LLD_STR.h       Generic Doubly Linked List for Strings.
                    v1.00  94-08-11
                    - Based on the LLD_BLOB module for variable size data-
                      items. Use the functions in the LLD module for
                      operations not specific to Strings.
                    - Refer to LLD_BLOB.h for comments.
                    - Note that ALL functions of LLD_BLOB are redefined
                      for consistency reasons.
                    This module has no associated .C files.
 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  |  \  ------------------------------------------------------------ */
#ifndef LLD_STR__H
#define LLD_STR__H
#include <string.h>
#include "lld_blob.h"
#define LLDstringCreate()       LLDblobCreate()
#define LLDstringInsert(l,s)    LLDblobInsert( l, s, strlen( s ) +1 )
#define LLDstringAdd(l,s)       LLDblobAdd( l, s, strlen( s ) +1 )
#define LLDstringPrepend(l,s)   LLDblobPrepend( l, s, strlen( s ) +1 )
#define LLDstringAppend(l,s)    LLDblobAppend( l, s, strlen( s ) +1 )
#define LLDstringDelete(l)      LLDblobDelete( l )
#define LLDstringData(l,d)      LLDblobData( l, d )
                                /* returns strlen() +1 !!! */
#endif /* LLD_STR__H */
/* ==== LLD_STR.h  end ================================================ */
```

TEXT STATISTICS

```
  1558 characters
    43 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    LLD_STR__H              21      22
    LLDblobAdd              31
    LLDblobAppend           35
    LLDblobCreate           27
    LLDblobData             39
    LLDblobDelete           37
    LLDblobInsert           29
    LLDblobPrepend          33
    LLDstringAdd            31
    LLDstringAppend         35
    LLDstringCreate         27
    LLDstringData           39
    LLDstringDelete         37
    LLDstringInsert         29
    LLDstringPrepend        33
    d       39+
    h       24
    l       29+     31+     33+     35+     37+     39+
    s       29+     31+     33+     35+
    string          24
    strlen          29      31      33      35
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLS.c           Generic Singly Linked Lists for fixed size data.
                    Each List has its own specific data size.
                    This version uses a dummy head node, which prevents
                    special handling of the first node.

                    v1.00  94-08-21
                    v1.01  95-10-21  Changed ListCountInit to unsigned.

                    Compile with NDEBUG not defined for debugging version.
                    Compile with NDEBUG defined for production version.

                    Prepared for use of a last node pointer.
                    Compile with USE_LASTPTR defined to use it.

                    The node pointers are restricted to valid values.
                    They point only in empty lists to invalid data.

                    Possible future enhancements:
                    - List(s) of free nodes for fast node memory alloc.
                      Or a special memory sub-allocator.
                    - FindFirst() & FindNext().
                    - Data access via first and/or last node pointers.
                      (duplicate the functions and change .current to
                      .first or .last)
                    - Node deletion via first and/or last node pointers.
                      (as for access functions, then simplify ...)

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#include <stdarg.h>         /* variable arg handling */
#include <assert.h>         /* debugging */

#if defined( __TURBOC__ ) && !defined( __BORLANDC__ )
 #include <stdio.h> /* required for early Turbo compilers with assert() */
#endif

#include "ll_defs.h"        /* debugging incl MALLOC (re-) definition   */
#include "LLS.h"            /* also includes extkword.h if necessary    */

#define NO_PROBLEMS LIST_NO_PROBLEMS /* local redefinition */

struct Node
{
    struct Node * next;
    int data;               /* also place-holder for larger items.      */
};                          /* actual nodes have various sizes,         */
                            /* but a fixed size within a list.          */
struct ListHead
{
    struct Node * current;  /* will actually point to preceding node    */
    struct Node * first;    /* always points to dummy head node         */

#ifdef USE_LASTPTR

    struct Node * last;     /* will actually point to preceding node    */

#endif

    int itemsize ;          /* zero value: used as 'list not used' flag */
};

#define ERR_MEMORY          -1

#define NODE_MALLOC(list)   (struct Node *) \
                            MALLOC( ListControl[ list ].itemsize \
                                    + sizeof( struct Node * ), char )

#define NODE_FREE(node)     FREE(node)

/* ---- Local data ---------------------------------------------------- */

static struct ListHead * ListControl = NULL;
static unsigned int ListCount ;

/* ---- LL system management ------------------------------------------ */

static int ListInit( int List, int ItemSize )
{
    struct Node * Tmp ;

    /* Create dummy head node.
       This is not a part of of the ListControl structure because that can
       and will move, making 'first' invalid. That _could_ be handled by
       adjusting it; or by getting rid of 'first' entirely and having its
       function taken over by "&.head" and ".first->next" by ".head.next".
    */
    if( 0 != ItemSize )
    {
        Tmp = NODE_MALLOC( List );
        if( NULL == Tmp )
        {
            return ERR_MEMORY ;
        }
        Tmp->next = NULL;
        Tmp->data = 0x4709 ; /* dummy value */
```

```
    }
    else
        Tmp = NULL ;

    /* initialize control structure
    */
    ListControl[ List ].current  =
    ListControl[ List ].first    = Tmp ;

#ifdef USE_LASTPTR

    ListControl[ List ].last     = Tmp ;

#endif

    ListControl[ List ].itemsize = ItemSize ; /* zero: list not used    */
    return NO_PROBLEMS ;
}

int LLSsystemInit( unsigned ListCountInit )
{
    assert( ( ListCountInit -1 ) <= 20 -1 );
                /* higher than 20 is ridiculous for an initial setup   */
                /* zero is useless                                     */

    if( NULL != ListControl )
    {
        return NO_PROBLEMS ; /* LL system already initialized */
    }

    ListControl = MALLOC( ListCountInit, struct ListHead );
    if( NULL == ListControl )
    {
        return ERR_MEMORY ;
    }

    for( ListCount = 0 ; ListCount < ListCountInit ; ListCount++ )
        ListInit( ListCount, 0 ); /* just mark it as usable ... */

    /* ListCount is now ListCountInit */
    assert( ListCount == ListCountInit );

    return NO_PROBLEMS;
}

int LLScreate( int ItemSize )
{
    unsigned List ;

    assert( (unsigned) ( ItemSize -1 ) < 1024 -1 ) ;
                            /* limit to 1kB. A size of 0 is ridiculous */

    /* trigger automatic system initialisation if necessary
    */
    if( NULL == ListControl  &&  0 != LLSsystemInit( 1 ))
    {
        return ERR_MEMORY ;
    }

    /* Look for empty slot
    */
    for( List = 0; List < ListCount; List++ )
    {
        if( 0 == ListControl[ List ].itemsize )
            break;
    }

    /*  What if NO EMPTY slot ???
    */
    if( List == ListCount )
    {
        struct ListHead * tmp ;         /* ListControl expansion needed */

        tmp = MALLOC( ListCount + 1, struct ListHead );
        if( NULL == tmp )
        {                               /* realloc is not used because  */
            return ERR_MEMORY ;         /* MALLOC is not necessarily    */
        }                               /* based on malloc()            */

        memcpy( tmp, ListControl, ListCount * sizeof( struct ListHead ));
        ListControl = tmp ;
        ListCount++ ;
    }

    /* create dummy head node and set up ListControl for the list.
    */
    if( ERR_MEMORY == ListInit( List, ItemSize ))
    {
        return ERR_MEMORY ;
    }

    return (int) List ;
}

void LLSdelete( int List )
{
    struct Node * Tmp ;
    struct Node * Old ;

    assert( (unsigned) List < ListCount );
```

```
201 |
202 |     Tmp = ListControl[ List ].first ; /* dummy head is also deleted !!! */
203 |     while( NULL != Tmp )              /* still assuming last node has   */
204 |     {                                 /* a NULL next pointer ...        */
205 |         Old = Tmp ;
206 |         Tmp = Old->next;
207 |         NODE_FREE( Old ); /* data already presumed to be deleted */
208 |     }
209 |
210 |     ListInit( List, 0 ); /* 0: mark list as not used. */
211 |
212 |     return ;
213 | }
214 |
215 | /* ---- LL system maintenance ---------------------------------------- */
216 |
217 | int LLScheck( int List )
218 | {
219 |     if( NULL == ListControl )
220 |     {
221 |         return LIST_SYSTEM_NULL ;
222 |     }
223 |
224 |     if( (unsigned) List >= ListCount )
225 |     {
226 |         return LIST_INV_NUMBER ;
227 |     }
228 |
229 |     if( 0 == ListControl[ List ].itemsize )
230 |     {
231 |         return LIST_NOT_CREATED ;
232 |     }
233 |
234 |     if( NULL == ListControl[ List ].first )
235 |     {
236 |         return LIST_ERR_HEAD ;
237 |     }
238 |
239 |     /* Validate current pointer,
240 |        and the last node pointer if it is used ...
241 |     */
242 |     if( NULL == ListControl[ List ].current )
243 |     {
244 |         return LIST_CORRUPT7 ;    /* shouldn't be NULL with a good head */
245 |     }
246 |
247 |     if( NULL != ListControl[ List ].first->next )       /* list empty ? */
248 |     {                                                   /* not empty    */
249 |         struct Node * tmp = ListControl[ List ].first ;
250 |
251 |         if( NULL == ListControl[ List ].current->next )
252 |         {
253 |             return LIST_CORRUPT6 ; /* a NULL next pointer is only valid */
254 |         }                          /* for an empty list.               */
255 |
256 |         /* look for .current in list
257 |         */
258 |         while( tmp != ListControl[ List ].current )
259 |         {
260 |             tmp = tmp->next ;
261 |
262 |             if( NULL == tmp )
263 |             {
264 |                 return LIST_CORRUPT5 ;  /* current not found in list */
265 |             }
266 |         }
267 |
268 | #ifdef USE_LASTPTR
269 |
270 |         /* Found .current in list.
271 |            Now look for valid last node pointer in list
272 |         */
273 |         if( NULL == ListControl[ List ].last )
274 |         {
275 |             return LIST_ERR_LAST ;
276 |         }
277 |
278 |         while( tmp != ListControl[ List ].last )
279 |         {
280 |             tmp = tmp->next ;
281 |             if( NULL == tmp )
282 |             {
283 |                 return LIST_CORRUPT4 ;  /* last not found in list */
284 |             }
285 |         }
286 |
287 |         /* Found .last in list but is it really the last ?
288 |            Note that last should actually point to the preceding node ...
289 |            Note: tmp == .last
290 |         */
291 |         if( NULL == tmp->next || NULL != tmp->next->next )
292 |         {
293 |             return LIST_CORRUPT3 ; /* a NULL next pointer is only valid */
294 |         }                          /* for an empty list. But next->next */
295 |                                    /* should be NULL for last pointer   */
296 | #endif
297 |
298 |         return NO_PROBLEMS ;
299 |     }
300 |
```

```
301 |     /* .first->next == NULL -> list is empty
302 |     */
303 |     if( ListControl[ List ].current != ListControl[ List ].first )
304 |     {
305 |         return LIST_CORRUPT2 ;
306 |     }
307 |
308 | #ifdef USE_LASTPTR
309 |
310 |     if( ListControl[ List ].last != ListControl[ List ].first )
311 |     {
312 |         return LIST_CORRUPT1 ;
313 |     }
314 |
315 | #endif
316 |
317 |     return LIST_EMPTY ;
318 | }
319 |
320 | /* ---- node management ------------------------------------------------ */
321 |
322 | int LLSnodeInsert( int List, ... )      /* insert _BEFORE_ current node */
323 | {
324 |     va_list DataPtr ;
325 |     int Retval ;
326 |
327 |     /* set DataPtr to the address of "..."
328 |        then action, cleanup and return.
329 |     */
330 |     va_start( DataPtr, List );
331 |
332 |     Retval = LLSnodeInsertFrom( List, DataPtr );
333 |
334 |     va_end( DataPtr );
335 |     return Retval ;
336 | }
337 |
338 | int LLSnodeAdd( int List, ... )          /* insert _AFTER_ current node */
339 | {
340 |     va_list DataPtr ;
341 |     int Retval ;
342 |
343 |     /* set DataPtr to the address of "..."
344 |        then action, cleanup and return.
345 |     */
346 |     va_start( DataPtr, List );
347 |
348 |     Retval = LLSnodeAddFrom( List, DataPtr );
349 |
350 |     va_end( DataPtr );
351 |     return Retval ;
352 | }
353 |
354 | int LLSnodePrepend( int List, ... )             /* insert as first node */
355 | {
356 |     va_list DataPtr ;
357 |     int Retval ;
358 |
359 |     /* set DataPtr to the address of "..."
360 |        then action, cleanup and return.
361 |     */
362 |     va_start( DataPtr, List );
363 |
364 |     Retval = LLSnodePrependFrom( List, DataPtr );
365 |
366 |     va_end( DataPtr );
367 |     return Retval ;
368 | }
369 |
370 | int LLSnodeAppend( int List, ... )              /* insert as last node */
371 | {
372 |     va_list DataPtr ;
373 |     int Retval ;
374 |
375 |     /* set DataPtr to the address of "..."
376 |        then action, cleanup and return.
377 |     */
378 |     va_start( DataPtr, List );
379 |
380 |     Retval = LLSnodeAppendFrom( List, DataPtr );
381 |
382 |     va_end( DataPtr );
383 |     return Retval ;
384 | }
385 |
386 | int LLSnodeInsertFrom( int List, void * Source )
387 | {                                       /* insert _BEFORE_ current node */
388 |     struct Node * New ;
389 |
390 |     assert( (unsigned) List < ListCount );
391 |
392 |     /* create new node if possible
393 |     */
394 |     New = NODE_MALLOC( List );
395 |     if( NULL == New )
396 |     {
397 |         return ERR_MEMORY ;
398 |     }
399 |
400 |     /* fill node with data and link to previous node
```

```
401 |        Note that explicitly changing .current is not necessary!
402 |     */
403 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
404 |     New->next = ListControl[ List ].current->next;
405 |     ListControl[ List ].current->next = New ;
406 |
407 | #ifdef USE_LASTPTR
408 |     /* advance last node pointer if needed
409 |     */
410 |     if( NULL != ListControl[ List ].last->next->next )
411 |         ListControl[ List ].last = ListControl[ List ].last->next ;
412 |
413 | #endif
414 |
415 |     return NO_PROBLEMS;
416 | }
417 |
418 | int LLSnodeAddFrom( int List, void * Source )
419 | {                                       /* insert _AFTER_ current node */
420 |     struct Node * New ;
421 |
422 |     assert( (unsigned) List < ListCount );
423 |
424 |     /* create new node if possible
425 |     */
426 |     New = NODE_MALLOC( List );
427 |     if( NULL == New )
428 |     {
429 |         return ERR_MEMORY ;
430 |     }
431 |
432 |     /* fill node with data and link to previous node,
433 |        with special handling if the list is empty
434 |        Note that the changing of .current is the only difference with
435 |        the LLSnodeInsertFrom() function!
436 |     */
437 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
438 |
439 |     if( NULL != ListControl[ List ].current->next )
440 |         ListControl[ List ].current = ListControl[ List ].current->next ;
441 |
442 |     New->next = ListControl[ List ].current->next;
443 |     ListControl[ List ].current->next = New;
444 |
445 | #ifdef USE_LASTPTR
446 |     /* advance last node pointer if needed
447 |     */
448 |     if( NULL != ListControl[ List ].last->next->next )
449 |         ListControl[ List ].last = ListControl[ List ].last->next ;
450 |
451 | #endif
452 |
453 |     return NO_PROBLEMS;
454 | }
455 |
456 | int LLSnodePrependFrom( int List, void * Source )
457 | {                                           /* insert as first node */
458 |     struct Node * New ;
459 |
460 |     assert( (unsigned) List < ListCount );
461 |
462 |     /* create new node if possible
463 |     */
464 |     New = NODE_MALLOC( List );
465 |     if( NULL == New )
466 |     {
467 |         return ERR_MEMORY ;
468 |     }
469 |
470 |     /* fill node with data
471 |     */
472 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
473 |     New->next = ListControl[ List ].first->next;
474 |     ListControl[ List ].first->next = New;
475 |
476 |     if( NULL != New->next && NULL == New->next->next )
477 |     {
478 |         /* The new node is not the only node and is the node preceding
479 |            the actual last node.
480 |            So it is the first of only two valid nodes.
481 |            Note that before insertion of 'New ' .current was .first
482 |            As was the optional last node pointer.
483 |            Note also that this section is a consequence of using a
484 |            'trailing' current node pointer!
485 |         */
486 |         ListControl[ List ].current = New ; /* == .current->next */
487 |
488 | #ifdef USE_LASTPTR
489 |
490 |         ListControl[ List ].last = New ; /* == .last->next */
491 |
492 | #endif
493 |
494 |     }
495 |
496 |     return NO_PROBLEMS;
497 | }
498 |
499 | int LLSnodeAppendFrom( int List, void * Source )
500 | {                                            /* insert as last node */
```

```
501 |     struct Node * New ;
502 |
503 |     assert( (unsigned) List < ListCount );
504 |
505 |     /* create new node if possible
506 |     */
507 |     New = NODE_MALLOC( List );
508 |     if( NULL == New )
509 |     {
510 |         return ERR_MEMORY ;
511 |     }
512 |
513 |     /* fill node with data
514 |     */
515 |     memcpy( & New->data, Source, ListControl[ List ].itemsize );
516 |     New->next = NULL ;
517 |
518 | #ifdef USE_LASTPTR
519 |
520 |     if( NULL != ListControl[ List ].last->next )    /* non empty list ? */
521 |         ListControl[ List ].last = ListControl[ List ].last->next ;
522 |     ListControl[ List ].last->next = New ;
523 |
524 | #else
525 |
526 |     {
527 |         struct Node * Tmp = ListControl[ List ].current;
528 |
529 |         while( NULL != Tmp->next ) /* find last node */
530 |             Tmp = Tmp->next ;
531 |
532 |         Tmp->next = New ;
533 |     }
534 |
535 | #endif
536 |
537 |     return NO_PROBLEMS;
538 | }
539 |
540 | void LLSnodeDelete( int List )
541 | {
542 |     {
543 |         struct Node * Old = ListControl[ List ].current->next ;
544 |
545 |         assert( (unsigned) List < ListCount );
546 |
547 |         if( NULL == ListControl[ List ].current->next )
548 |         {
549 |             return ;  /* nothing there to delete ... */
550 |         }
551 |
552 |         ListControl[ List ].current->next = Old->next ;
553 |         NODE_FREE( Old );
554 |     }
555 |
556 |     /* Modification to prevent current and last node pointers pointing
557 |        past the last node of a list: adjust the current and last node
558 |        pointers when the last node was deleted
559 |     */
560 |     if( NULL == ListControl[ List ].current->next )
561 |     {
562 |         struct Node * Tmp = ListControl[ List ].first;
563 |
564 |         if( NULL != Tmp->next )                 /* list not empty ?     */
565 |             while( NULL != Tmp->next->next )    /* find the node before */
566 |                 Tmp = Tmp->next ;               /* the last node        */
567 |
568 |         ListControl[ List ].current = Tmp ;
569 |
570 | #ifdef USE_LASTPTR
571 |
572 |         ListControl[ List ].last = Tmp ;
573 |
574 | #endif
575 |
576 |     }
577 |     return ;
578 | }
579 |
580 | int LLSnodeFind( int List, CompFunPtr Compare, void * DataPtr )
581 | {                       /* FindFirst/FindNext format may be needed ... */
582 |     int RetVal ;
583 |
584 |     assert( (unsigned) List < ListCount );
585 |
586 |     if( NULL == ListControl[ List ].first->next )
587 |     {
588 |         return 2; /* a compare usually returns just -1, 0 or 1 !!! */
589 |     }
590 |
591 |     if( NULL == Compare ) /* default to memcmp with .itemsize */
592 |     {
593 |         while( 0 != (RetVal = memcmp( DataPtr,
594 |                                 & ListControl[ List ].current->next->data,
595 |                                 ListControl[ List ].itemsize ))
596 |                && NULL != ListControl[ List ].current->next->next )
597 |         {
598 |             ListControl[ List ].current=ListControl[ List ].current->next;
599 |         }
600 |         return RetVal ;
```

```
601 |     }
602 |     else
603 |     {
604 |         while( 0 != (RetVal = (*Compare)( DataPtr,
605 |                             & ListControl[ List ].current->next->data ))
606 |                && NULL != ListControl[ List ].current->next->next )
607 |         {
608 |             ListControl[ List ].current=ListControl[ List ].current->next;
609 |         }
610 |         return RetVal ;
611 |     }
612 | }
613 |
614 | /* ---- current node pointer management ------------------------------ */
615 |
616 | int  LLSnodePtr2First( int List )
617 | {
618 |     assert( (unsigned) List < ListCount );
619 |
620 |     ListControl[ List ].current = ListControl[ List ].first ;
621 |
622 |     return NULL != ListControl[ List ].first->next ;
623 | }
624 |
625 | int  LLSnodePtr2Last( int List )
626 | {
627 |     assert( (unsigned) List < ListCount );
628 |
629 | #ifdef USE_LASTPTR
630 |
631 |     ListControl[ List ].current = ListControl[ List ].last ;
632 |
633 |     return NULL != ListControl[ List ].first->next ;
634 |
635 | #else
636 |
637 |     /* special handling for empty list
638 |     */
639 |     if( NULL == ListControl[ List ].first->next )
640 |     {
641 |         return 0; /* .current also same as .first */
642 |     }
643 |
644 |     /* Let the current node pointer point to the last valid node
645 |     */
646 |     while( NULL != ListControl[ List ].current->next->next )
647 |         ListControl[ List ].current = ListControl[ List ].current->next ;
648 |
649 |     return 1;
650 |
651 | #endif
652 |
653 | }
654 |
655 | int  LLSnodePtr2Next( int List )
656 | {
657 |     assert( (unsigned) List < ListCount );
658 |
659 |     if( NULL == ListControl[ List ].current->next       /* empty list ? */
660 |         || NULL == ListControl[ List ].current->next->next ) /* at end ?*/
661 |     {
662 |         return 0 ;             /* note: 'at end' condition added to     */
663 |     }                          /* to prevent .current pointing past end */
664 |
665 |     ListControl[ List ].current = ListControl[ List ].current->next ;
666 |     return 1 ;
667 | }
668 |
669 | int  LLSnodePtr2Prev( int List )
670 | {
671 |     struct Node * Prev ;
672 |
673 |     assert( (unsigned) List < ListCount );
674 |
675 |     Prev = ListControl[ List ].first ;
676 |     if( NULL == Prev->next                       /* empty list or */
677 |         || ListControl[ List ].current == Prev ) /* at beginning? */
678 |     {
679 |         return 0 ;
680 |     }
681 |
682 |     /* Find previous Node
683 |     */
684 |     while( Prev->next != ListControl[ List ].current )
685 |         Prev = Prev->next ;
686 |
687 |     ListControl[ List ].current = Prev ;
688 |
689 |     return 1 ;
690 | }
691 |
692 | /* ---- stored data management -------------------------------------- */
693 |
694 | int LLSnodeInt( int List )
695 | {
696 |     return ListControl[ List ].current->next->data;
697 | }
698 |
699 | long LLSnodeLong( int List )
700 | {
```

```
701 |     return *((long *) &ListControl[ List ].current->next->data );
702 | }
703 |
704 | void * LLSnodePtr( int List )
705 | {
706 |     return *((void **) &ListControl[ List ].current->next->data );
707 | }
708 |
709 | void FAR * LLSnodeFptr( int List )
710 | {
711 |     return *((void FAR **) &ListControl[ List ].current->next->data );
712 | }
713 |
714 | int LLSnodeDataTo( int List, void * Destination )
715 | {
716 |     if( NULL != Destination )
717 |     {
718 |         memcpy( Destination,
719 |                 & ListControl[ List ].current->next->data,
720 |                 ListControl[ List ].itemsize );
721 |     }
722 |
723 |     return ListControl[ List ].itemsize ;       /* size needed for blob */
724 | }
725 |
726 | /* ==== LLS.c  end ================================================== */
```

TEXT STATISTICS

```
 20352 characters
   726 lines
```

LEXICAL STATISTICS

```
   106 comments [std-C]
     0 comments [C++]
    33 preprocessor instructions
     0 constants [character]
     2 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    CompFunPtr              580
    Compare         580     591     604
    DataPtr         324     330     332     334     340     346     348     350     356     362     364
          366       372     378     380     382     580     593     604
    Destination             714     716     718
    ERR_MEMORY               67      97     134     157     177     187     189     397     429     467
          510
    FAR             709     711
    FREE             73
    ItemSize         82      92     116     146     150     187
    LIST_CORRUPT1           312
    LIST_CORRUPT2           305
    LIST_CORRUPT3           293
    LIST_CORRUPT4           283
    LIST_CORRUPT5           264
    LIST_CORRUPT6           253
    LIST_CORRUPT7           244
    LIST_EMPTY              317
    LIST_ERR_HEAD           236
    LIST_ERR_LAST           275
    LIST_INV_NUMBER         226
    LIST_NOT_CREATED        231
    LIST_NO_PROBLEMS         45
    LIST_SYSTEM_NULL        221
    LLScheck        217
    LLScreate       146
    LLSdelete       195
    LLSnodeAdd              338
    LLSnodeAddFrom          348     418
    LLSnodeAppend           370
    LLSnodeAppendFrom       380     499
    LLSnodeDataTo           714
    LLSnodeDelete           540
    LLSnodeFind             580
    LLSnodeFptr             709
    LLSnodeInsert           322
    LLSnodeInsertFrom       332     386
    LLSnodeInt              694
    LLSnodeLong             699
    LLSnodePrepend          354
    LLSnodePrependFrom              364     456
    LLSnodePtr              704
    LLSnodePtr2First        616
    LLSnodePtr2Last         625
    LLSnodePtr2Next         655
    LLSnodePtr2Prev         669
    LLSsystemInit           120     155
    List             82      94     107     108     112     116     148     162+    164     170     187
          192       195     200     202     210     217     224     229     234     242     247     249
          251       258     273     278     303+    310+    322     330     332     338     346     348
          354       362     364     370     378     380     386     390     394     403     404     405
          410       411+    418     422     426     437     439     440+    442     443     448     449+
          456       460     464     472     473     474     486     490     499     503     507     515
          520       521+    522     527     540     543     545     547     552     560     562     568
          572       580     584     586     594     595     596     598+    605     606     608+    616
          618       620+    622     625     627     631+    633     639     646     647+    655     657
          659       660     665+    669     673     675     677     684     687     694     696     699
          701       704     706     709     711     714     719     720     723
    ListControl              70      77     107     108     112     116     126     131     132     155
          164       180     181     202     219     229     234     242     247     249     251     258
          273       278     303+    310+    403     404     405     410     411+    437     439     440+
          442       443     448     449+    472     473     474     486     490     515     520     521+
          522       527     543     547     552     560     562     568     572     586     594     595
          596       598+    605     606     608+    620+    622     631+    633     639     646     647+
          659       660     665+    675     677     684     687     696     701     706     711     719
          720       723
    ListCount        78     137+    138     141     162     170     174     180     182     200     224
          390       422     460     503     545     584     618     627     657     673
    ListCountInit           120     122     131     137     141
    ListHead         53      77     131     172     174     180
    ListInit         82     138     187     210
    MALLOC           70     131     174
    NODE_FREE        73     207     553
    NODE_MALLOC              69      94     394     426     464     507
    NO_PROBLEMS              45     117     128     143     298     415     453     496     537
    NULL             77      95      99     103     126     132     155     175     203     219     234
          242       247     251     262     273     281     291+    395     410     427     439     448
          465       476+    508     516     520     529     547     560     564     565     586     591
          596       606     622     633     639     646     659     660     676     716
    New             388     394     395     403     404     405     420     426     427     437     442
          443       458     464     465     472     473     474     476+    486     490     501     507
          508       515     516     522     532
    Node             47      49      55      56      60      69      71      84     197     198     249
          388       420     458     501     527     543     562     671
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLS.h           Generic Singly Linked List for fixed size data-items.

                    v1.00  94-08-21
                    v1.01  95-10-21  Changed ListCountInit to unsigned.

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#ifndef LLS__H
#define LLS__H

#ifndef FAR
#include "extkword.h"
#endif

#ifndef LL__ERR_H
#define LL__ERR_H       /* same values used in LLD ...                  */

enum ListErrors         /* return values for LLScheck()                 */
{                       /* The highest value is returned                */

    LIST_NO_PROBLEMS,   /* All is OK (multiple use)                     */
    LIST_EMPTY,         /* No data available                            */
    LIST_ERRORS,        /* Dummy to separate warnings from              */
                        /* ---- REAL errors ------------------------- */
    LIST_CORRUPT1,      /* invalid last node pointer: Not NULL          */
                        /*      (empty list: ptr should have been NULL) */
    LIST_CORRUPT2,      /* invalid current node pointer: Not NULL       */
                        /*      (empty list: ptr should have been NULL) */
    LIST_CORRUPT3,      /* invalid last node pointer: Not really last.  */
    LIST_CORRUPT4,      /* invalid last node pointer: Not in list.      */
    LIST_ERR_LAST,      /* invalid last node pointer: NULL              */
    LIST_CORRUPT5,      /* invalid current node pointer: Not in list.   */
    LIST_CORRUPT6,      /* invalid current->next node pointer: NULL     */
                        /*      although the list is not empty          */
    LIST_CORRUPT7,      /* NULL current node pointer                    */
    LIST_ERR_HEAD,      /* NULL first node pointer                      */
    LIST_NOT_CREATED,   /* List deleted or not created                  */
    LIST_INV_NUMBER,    /* List number out of range                     */
    LIST_SYSTEM_NULL,   /* List system not initialized                  */
};
            /* NOTE: 'invalid last node pointer' errors will ONLY occur */
            /* when LLS.c is compiled with USE_LASTPTR defined ...      */

typedef int (*CompFunPtr)( const void *, const void * );
                                            /* simplifies declarations */
#endif

/* ---- LL system management and maintenance -------------------------- */
int  LLSsystemInit( unsigned ListCount );
               /* returns -1 on failure. It is not required to call it. */
               /* A second call does nothing: ListCount is ignored.     */

int  LLScreate( int ItemSize );
                       /* returns list number to use or -1 on failure. */
                       /* MUST be called before using a list.          */
                       /* Calls LLsystemInit if necessary.             */

void LLSdelete( int List ); /* delete entire list, data is NOT free()'d */

int  LLScheck( int List ); /* returns enum ListErrors value             */
                           /* its primary purpose is debugging.         */

/* ---- Node management ----------------------------------------------
   Functions changing current node pointer to the new node.
   Each created list has its own -- fixed -- datasize. See LLcreate().
   An ellipsis "..." indicates the data to insert.
*/
int  LLSnodeInsert( int List, ... );      /* insert BEFORE current node */
int  LLSnodeAdd( int List, ... );         /* insert AFTER current node  */
         /* a return value of -1 indicates a memory allocation problem. */

/* Functions NOT changing the current node pointer.
   Especially intended for implementation of Queue's and Stacks.
*/
int  LLSnodePrepend( int List, ... );           /* insert as first node */
int  LLSnodeAppend( int List, ... );            /* insert as last node  */
         /* a return value of -1 indicates a memory allocation problem. */

/* The following four functions are essentially the same as the preceding
   four. The data is however not passed by value but by reference.
*/
int  LLSnodeInsertFrom( int List, void * Source );
int  LLSnodeAddFrom( int List, void * Source );
int  LLSnodePrependFrom( int List, void * Source );
int  LLSnodeAppendFrom( int List, void * Source );

void LLSnodeDelete( int List );                  /* remove current node */
        /* current node ptr moved to next node. UNLESS the deleted node */
        /* was the last node: then current ptr moved to previous node   */

int  LLSnodeFind( int List, CompFunPtr Compare, void * DataPtr );
        /* Find *DataPtr in the List using the *Compare function.       */
        /* Returns the return value of *Compare. 0 == equal == found.   */
        /* non-zero == not found. Current node is set to found node.    */
        /* Returns 2 for an empty list.                                 */
```

```
101 |         /* First checked node is current node.                          */
102 |         /* A NULL Compare-function defaults to the use of memcmp() with */
103 |         /* the list's itemsize as third (size) parameter.               */
104 |         /* simple implementation. FindFirst and FindNext may be needed. */
105 |
106 | /* ---- current node pointer management ---------------------------------
107 |    These functions change the current node pointer and return 1 when there
108 |    was a change. A return of 0 indicates trying to move past the begin or
109 |    the end of the list, or an empty list. The return value is intended for
110 |    iteration purposes. I.e. stopping a scan through a list.
111 | */
112 | int  LLSnodePtr2First( int List );
113 | int  LLSnodePtr2Last( int List );
114 | int  LLSnodePtr2Next( int List );
115 | int  LLSnodePtr2Prev( int List );
116 |
117 | /* ---- stored data management -------------------------------------------
118 |    return typed data:
119 | */
120 | int  LLSnodeInt( int List );
121 | long LLSnodeLong( int List );
122 | void * LLSnodePtr( int List );
123 | void FAR * LLSnodeFptr( int List );
124 |
125 | /* 'return' typeless data. The return value is the size of the data.
126 |    The data is transferred to Destination.
127 |    If 'Destination' is NULL, the only action is returning the size.
128 | */
129 | int LLSnodeDataTo( int List, void * Destination );
130 |
131 | #endif /* LLS__H */
132 | /* ==== LLS.h  end ================================================== */
```

TEXT STATISTICS

```
  6534 characters
   132 lines
```

LEXICAL STATISTICS

```
    61 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CompFunPtr               49      96
    Compare          96
    DataPtr          96
    Destination             129
    FAR              16     123
    ItemSize         58
    LIST_CORRUPT1            30
    LIST_CORRUPT2            32
    LIST_CORRUPT3            34
    LIST_CORRUPT4            35
    LIST_CORRUPT5            37
    LIST_CORRUPT6            38
    LIST_CORRUPT7            40
    LIST_EMPTY               27
    LIST_ERRORS              28
    LIST_ERR_HEAD            41
    LIST_ERR_LAST            36
    LIST_INV_NUMBER          43
    LIST_NOT_CREATED         42
    LIST_NO_PROBLEMS         26
    LIST_SYSTEM_NULL         44
    LLS__H           13      14
    LLScheck         65
    LLScreate        58
    LLSdelete        63
    LLSnodeAdd               74
    LLSnodeAddFrom           88
    LLSnodeAppend            81
    LLSnodeAppendFrom        90
    LLSnodeDataTo           129
    LLSnodeDelete            92
    LLSnodeFind              96
    LLSnodeFptr             123
    LLSnodeInsert            73
    LLSnodeInsertFrom        87
    LLSnodeInt              120
    LLSnodeLong             121
    LLSnodePrepend           80
    LLSnodePrependFrom               89
    LLSnodePtr              122
    LLSnodePtr2First        112
    LLSnodePtr2Last         113
    LLSnodePtr2Next         114
    LLSnodePtr2Prev         115
    LLSsystemInit            54
    LL__ERR_H        20      21
    List             63      65      73      74      80      81      87      88      89      90      92
            96     112     113     114     115     120     121     122     123     129
    ListCount        54
    ListErrors               23
    Source           87      88      89      90
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLS_BLOB.c      Generic Singly Linked Lists for Binary Large OBjects.
                    Linked Lists for variable size data-items.
                    This is a 'front end' for the generic LLS module.

                    v1.00  94-08-11

    _____           This version is Public Domain.
   /_|__|           A.Reitsma, Delft, The Netherlands.
  /  |  \  ------------------------------------------------------------ */

#include "ll_defs.h"
#include "lls.h"            /* the generic LLS functions ...            */
#include "lls_blob.h"       /* also includes extkword.h if necessary    */

struct BlobDesc
{
    void * data ;           /* 'data' can be obtained by LLSnodePtr() ! */
    unsigned size ;
};

#define ERR_MEMORY          -1

/* ---- LL blob system management ------------------------------------- */

int LLSblobCreate( void )
{
    return LLScreate( sizeof( struct BlobDesc ));
}

/* ---- LL blob node management --------------------------------------- */

int LLSblobInsert( int List, void * Source, unsigned Size )
{                                   /* insert _BEFORE_ current node */
    struct BlobDesc Blob ;

    Blob.size = Size ;
    Blob.data = malloc( Size );

    if( NULL == Blob.data )
    {
        return ERR_MEMORY ;
    }

    memcpy( Blob.data, Source, Size );
    LLSnodeInsertFrom( List, & Blob );

    return LIST_NO_PROBLEMS ;
}

int LLSblobAdd( int List, void * Source, unsigned Size )
{                                    /* insert _AFTER_ current node */
    struct BlobDesc Blob ;

    Blob.size = Size ;
    Blob.data = malloc( Size );

    if( NULL == Blob.data )
    {
        return ERR_MEMORY ;
    }

    memcpy( Blob.data, Source, Size );
    LLSnodeAddFrom( List, & Blob );

    return LIST_NO_PROBLEMS ;
}

int LLSblobPrepend( int List, void * Source, unsigned Size )
{                                              /* insert as first node */
    struct BlobDesc Blob ;

    Blob.size = Size ;
    Blob.data = malloc( Size );

    if( NULL == Blob.data )
    {
        return ERR_MEMORY ;
    }

    memcpy( Blob.data, Source, Size );
    LLSnodePrependFrom( List, & Blob );

    return LIST_NO_PROBLEMS ;
}

int LLSblobAppend( int List, void * Source, unsigned Size )
{                                               /* insert as last node */
    struct BlobDesc Blob ;

    Blob.size = Size ;
    Blob.data = malloc( Size );

    if( NULL == Blob.data )
    {
        return ERR_MEMORY ;
    }

```

```
101 |     memcpy( Blob.data, Source, Size );
102 |     LLSnodeAppendFrom( List, & Blob );
103 |
104 |     return LIST_NO_PROBLEMS ;
105 | }
106 |
107 | void LLSblobDelete( int List )
108 | {
109 |     struct BlobDesc Blob ;
110 |
111 |     LLSnodeDataTo( List, & Blob );
112 |     free( Blob.data );
113 |
114 |     LLSnodeDelete( List );
115 |
116 |     return ;
117 | }
118 |
119 | /* ---- stored data management ----------------------------------------- */
120 |
121 | unsigned LLSblobData( int List, void * Destination )
122 | {
123 |     struct BlobDesc Blob ;
124 |
125 |     LLSnodeDataTo( List, & Blob );
126 |
127 |     if( NULL != Destination )
128 |         memcpy( Destination, Blob.data, Blob.size );
129 |
130 |     return Blob.size ;       /* size needed for blob */
131 | }
132 |
133 | /* ==== LLS_BLOB.c  end ================================================ */
```

TEXT STATISTICS

```
  3404 characters
   133 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    Blob            37      39      40      42      47      48      55      57      58      60      65
           66       73      75      76      78      83      84      91      93      94      96     101
          102     109     111     112     123     125     128+    130
    BlobDesc        18      30      37      55      73      91     109     123
    Destination            121     127     128
    ERR_MEMORY              24      44      62      80      98
    LIST_NO_PROBLEMS        50      68      86     104
    LLSblobAdd              53
    LLSblobAppend           89
    LLSblobCreate           28
    LLSblobData            121
    LLSblobDelete          107
    LLSblobInsert           35
    LLSblobPrepend          71
    LLScreate       30
    LLSnodeAddFrom          66
    LLSnodeAppendFrom      102
    LLSnodeDataTo          111     125
    LLSnodeDelete          114
    LLSnodeInsertFrom       48
    LLSnodePrependFrom              84
    List            35      48      53      66      71      84      89     102     107     111     114
          121     125
    NULL            42      60      78      96     127
    Size            35      39      40      47      53      57      58      65      71      75      76
           83       89      93      94     101
    Source          35      47      53      65      71      83      89     101
    data            20      40      42      47      58      60      65      76      78      83      94
           96     101     112     128
    free           112
    malloc          40      58      76      94
    memcpy          47      65      83     101     128
    size            21      39      57      75      93     128     130
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LLS_BLOB.h      Generic Singly Linked List for Binary Large OBjects.
                    Linked Lists for variable size data-items.

                    v1.00  94-08-11

                    - Based on the LLS module for fixed size data-items.
                    - Use the functions in the LLS module for operations
                      not specific to Blobs. You can use LLSnodePtr() to
                      obtain a pointer to the stored Blob.
                    - Note that From and To suffixes to function names are
                      implied in the Blob data related functions.

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#ifndef LLS_BLOB__H
#define LLS_BLOB__H

/* ---- LL blob system management and maintenance --------------------- */

int  LLSblobCreate( void );
                        /* returns list number to use or -1 on failure. */
                        /* MUST be called before using a list of blobs. */

/* ---- Node management ----------------------------------------------
   Functions changing current node pointer to the new node.
   A return value of -1 indicates a memory allocation problem.
*/
int  LLSblobInsert( int List, void * Source, unsigned Size );
                                          /* insert BEFORE current node */
int  LLSblobAdd( int List, void * Source, unsigned Size );
                                          /* insert AFTER current node  */

/* Functions NOT changing the current node pointer.
   Especially intended for implementation of Queue's and Stacks.
*/
int  LLSblobPrepend( int List, void * Source, unsigned Size );
                                                /* insert as first node */
int  LLSblobAppend( int List, void * Source, unsigned Size );
                                                /* insert as last node  */

void LLSblobDelete( int List );
        /* remove current node and free() the data.                    */
        /* current node ptr moved to next node. UNLESS the deleted node */
        /* was the last node: then current ptr moved to previous node   */

/* ---- stored data management ---------------------------------------
   'return' typeless data. The return value is the size of the data.
   The data is transferred to Destination.
   If 'Destination' is NULL, the only action is returning the size.
*/
unsigned LLSblobData( int List, void * Destination );

#endif /* LLS_BLOB__H */
/* ==== LLS_BLOB.h  end ================================================ */
```

TEXT STATISTICS

```
  2788 characters
    59 lines
```

LEXICAL STATISTICS

```
    17 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

Destination		56				
LLS_BLOB__H		20	21			
LLSblobAdd		35				
LLSblobAppend		43				
LLSblobCreate		25				
LLSblobData		56				
LLSblobDelete		46				
LLSblobInsert		33				
LLSblobPrepend		41				
List	33	35	41	43	46	56
Size	33	35	41	43		
Source	33	35	41	43		

```
/* +++Date last modified: 05-Jul-1997 */
/*
/* ======================================================================
    LLS_STR.h       Generic Singly Linked List for Strings.
/*
                    v1.00  94-08-11
/*
                    - Based on the LLS_BLOB module for variable size data-
                      items. Use the functions in the LLS module for
                      operations not specific to Strings.
                    - Refer to LLS_BLOB.h for comments.
                    - Note that ALL functions of LLS_BLOB are redefined
                      for consistency reasons.
/*
                    This module has no associated .C files.
/*
 _____              This version is Public Domain.
/_|__|              A.Reitsma, Delft, The Netherlands.
 | \  ------------------------------------------------------------ */
/*
#ifndef LLS_STR__H
#define LLS_STR__H
/*
#include <string.h>
#include "lls_blob.h"
/*
#define LLSstringCreate()       LLSblobCreate()
/*
#define LLSstringInsert(l,s)    LLSblobInsert( l, s, strlen( s ) +1 )
/*
#define LLSstringAdd(l,s)       LLSblobAdd( l, s, strlen( s ) +1 )
/*
#define LLSstringPrepend(l,s)   LLSblobPrepend( l, s, strlen( s ) +1 )
/*
#define LLSstringAppend(l,s)    LLSblobAppend( l, s, strlen( s ) +1 )
/*
#define LLSstringDelete(l)      LLSblobDelete( l )
/*
#define LLSstringData(l,d)      LLSblobData( l, d )
                                /* returns strlen() +1 !!! */
/*
#endif /* LLS_STR__H */
/* ==== LLS_STR.h  end ================================================ */
```

TEXT STATISTICS

```
  1558 characters
    43 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    LLS_STR__H              21      22
    LLSblobAdd              31
    LLSblobAppend           35
    LLSblobCreate           27
    LLSblobData             39
    LLSblobDelete           37
    LLSblobInsert           29
    LLSblobPrepend          33
    LLSstringAdd            31
    LLSstringAppend         35
    LLSstringCreate         27
    LLSstringData           39
    LLSstringDelete         37
    LLSstringInsert         29
    LLSstringPrepend        33
    d       39+
    h       24
    l       29+     31+     33+     35+     37+     39+
    s       29+     31+     33+     35+
    string          24
    strlen          29      31      33      35
```

```
/* +++Date last modified: 05-Jul-1997 */
|
/*  ======================================================================
|     DEFINES.h       Standard definitions etc.
|                     For simplification or for debugging substitution.
|
|                     v1.02  94-08-11  Stripped version.
|
|  _____              This version is Public Domain.
|  /_|__|             A.Reitsma, Delft, Nederland.
| /  | \  ------------------------------------------------------------ */
|
| #ifndef LL_DEFS__H
| #define LL_DEFS__H
|
| #include <stdlib.h>         /* for malloc() prototype */
| #include <string.h>         /* for memcpy() prototype */
|
| #define MALLOC(size,type)   (type *) malloc( (size) * sizeof( type ))
| #define FREE(mem)           free( mem )
| #define CALLOC(size,type)   (type *) calloc( (size), sizeof( type))
|
| #endif /* LL_DEFS__H */
| /*  === DEFINES.h end ============================================== */
```

TEXT STATISTICS

```
   922 characters
    24 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CALLOC          21
    FREE            20
    LL_DEFS__H              13      14
    MALLOC          19
    calloc          21
    free            20
    h       16      17
    malloc          19
    mem             20+
    size            19+     21+
    stdlib          16
    string          17
    type            19+     21+
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LL_DEMO.C   Demonstration of the use of my Linked List modules.
                Intentionally over-commented w.r.t the List stuff and
                under-commented w.r.t. the rest.
                Not all LLS functions are used, especially LLSnodeDelete()
                is not used. But I've tried to use all the most important
                functions. Even when this resulted in less than optimal
                functionality. E.g. the use of LLSnodeInsert for inserting
                the data for a directory in the list.

                All subdirectories in the current directory are displayed
                first sorted by name. Then all files sorted by date and
                time.
                The sorting is done by inserting a new node at the
                appropriate point in the list. This is not the most
                efficient way to sort, but it is the only possibility with
                these Linked Lists because some info internal to the list
                is (intentionally) not available to the user. Some kind of
                tree would be better anyway ;-)

                Doubly Linked Lists should work the same here as the used
                Singly Linked Lists. Just replace any 'LLS' by 'LLD'
                preserving the case. I haven't tried it though, as their
                only advantage is speed (at the expense of space) when
                moving backwards in a list.

                v1.00  97-05-22  Initial version.
                       97-05-23  Comments added and completed.

 _____          This version is Public Domain.
 /_|__|         A.Reitsma, Delft, Nederland.
/  | \  ------------------------------------------------------------ */

#include <stdio.h>        /* for printf()                          */
#include "dirport.h"      /* for all the FIND_* and ff_* stuff     */
#include "sniptype.h"     /* for Success_                          */
#include "lls.h"          /* the Singly Linked List stuff          */

/*  Comparison functions which are to be called by LLSnodeFind
*/
int DirComp( const void * D1, const void * D2 );
int FileComp( const void * D1, const void * D2 );

/*  Separate output functions for directories and files.
    Could be one function with a conditional on ff_attr() == _A_SUBDIR
*/
void DirOutput( int List );
void FileOutput( int List );

int CompareResult; /* Kludge to solve an (intentional) limitation of    */
                   /* the Linked List system. Serves as secondary       */
                   /* output channel from the ...Comp functions to main.*/

int main( int argc , char * argv[] )
{
    DOSFileData Info;

    /*  Create and initialize two lists
    */
    int ListDirs  = LLScreate( sizeof( DOSFileData ));
    int ListFiles = LLScreate( sizeof( DOSFileData ));

    if( Success_ != FIND_FIRST( argc < 2 ? "*.*" : argv[ 1 ],
                                _A_ANY, &Info ))
    {
        return EXIT_FAILURE;
    }

    do
    {
        if( _A_SUBDIR & ff_attr( &Info ))
        {
            /*  It is a directory but we'll ignore the '.' and '..' dirs
            */
            if( '.' == ff_name( &Info )[ 0 ])
                continue;

            /*  Search from the start ...
            */
            LLSnodePtr2First( ListDirs );
            /*  ... and set the 'current node pointer' to the node
                matching -- according to the function DirComp --
                the new data.
            */
            LLSnodeFind( ListDirs, DirComp, &Info );
            /*  LLSnodeFind has a problem in that it is basically designed
                to find EXACT matches. And DirComp says it is a match when
                it is the current node is 'larger' than the new data.
                Also -- unless the list is empty -- the design of the list
                system is such that the current node pointer can NEVER
                point at an empty (dummy) node. This means we can NEVER
                use LLSnodeInsert() to insert a node as the last element
                of the list.
                Resulting in the following if/else construct.
            */
            if( 0 <= CompareResult )
                LLSnodeInsert( ListDirs, Info );
            else
```

```
101 |                 LLSnodeAdd( ListDirs, Info );
102 |             /*  Now note that Info is put directly on the stack. This
103 |                 _should_ cause a warning, but you can safely ignore it.
104 |             */
105 |         }
106 |         else if( _A_VOLID & ff_attr( &Info ))
107 |         {
108 |             /*  Do nothing with volume label */
109 |         }
110 |         else
111 |         {   /*  It is a file. Deal with it in a similar way as a
112 |                 directory. The major difference is the way the data
113 |                 is inserted in the list. Note that the difference is
114 |                 for demonstration purposes!
115 |             */
116 |             LLSnodePtr2First( ListFiles );
117 |             LLSnodeFind( ListFiles, FileComp, &Info );
118 |             if( 0 <= CompareResult )
119 |                 LLSnodeInsertFrom( ListFiles, &Info );
120 |             else
121 |                 LLSnodeAddFrom( ListFiles, &Info );
122 |             /*  In this case a _pointer_ to the info is put on the stack.
123 |                 This is -- in this case -- more efficient.
124 |                 With data sizes less than the size of a pointer the direct
125 |                 method -- the way a directory is handled earlier -- is
126 |                 both faster and smaller.
127 |             */
128 |         }
129 |     }while( Success_ == FIND_NEXT( &Info ));
130 |
131 |     FIND_END( &Info );
132 |
133 |     /*  Output the content of the List with directories,
134 |         then get rid of it.
135 |     */
136 |     DirOutput( ListDirs);
137 |     LLSdelete( ListDirs );
138 |
139 |     /*  And similar for the files.
140 |     */
141 |     FileOutput( ListFiles );
142 |     LLSdelete( ListFiles );
143 |
144 |     return EXIT_SUCCESS;
145 | }
146 |
147 | int DirComp( const void * D1, const void * D2 )
148 | {
149 |     int RetVal = strcmp( ff_name( (DOSFileData *) D2 ),
150 |                          ff_name( (DOSFileData *) D1 ));
151 |
152 |     CompareResult = RetVal;
153 |
154 |     /*  Return a 'match' (0) if the item in the list is 'larger' than the
155 |         new data.
156 |     */
157 |     return RetVal >= 0 ? 0 : 1 ;
158 | }
159 |
160 | int FileComp( const void * D1, const void * D2 )
161 | {
162 |     /*  Date and time structs are transformed into 'normal' ints by
163 |         casting. This is not really portable to non-DOS systems ...
164 |     */
165 |     int RetVal = *((int *) & ff_date( (DOSFileData *) D2 )) -
166 |                  *((int *) & ff_date( (DOSFileData *) D1 ));
167 |
168 |     if( 0 == RetVal )
169 |         RetVal = *((int *) & ff_time( (DOSFileData *) D2 )) -
170 |                  *((int *) & ff_time( (DOSFileData *) D1 ));
171 |
172 |     CompareResult = RetVal;
173 |
174 |     /*  Return a 'match' (0) if the item in the list is 'larger' than the
175 |         new data.
176 |     */
177 |     return RetVal >= 0 ? 0 : 1 ;
178 | }
179 |
180 | void DirOutput( int List )
181 | {
182 |     /*  In this case we KNOW the type of the data for certain.
183 |         Otherwise we'd use a pointer and malloc sufficient memory.
184 |         E.g. by:
185 |             char * Sufficient = malloc( LLSnodeDataTo( List, NULL );
186 |         Note the NULL! It says: "return only the size".
187 |         Later on we'd use "Sufficient" in stead of "&Data".
188 |     */
189 |     DOSFileData Data;
190 |
191 |     /*  From the start of the list ...
192 |     */
193 |     if( ! LLSnodePtr2First( List ))
194 |     {
195 |         return; /* list is empty */
196 |     }
197 |
198 |     do
199 |     {
200 |         /* ... transfer the data the current node points at to 'Data' ...
```

```
201 |         */
202 |         LLSnodeDataTo( List, &Data );
203 |         /*  ... then output it.
204 |         */
205 |         printf( "%-14.14s  %10.10s  "
206 |                 "%4.4d-%2.2d-%2.2d  "
207 |                 "%2.2d:%2.2d:%2.2d  "
208 |                 "%2.2x\n",
209 |                 ff_name( &Data ), "<<dir>>",
210 |                 1980 + ff_yr( &Data ), ff_mo( &Data ), ff_day( &Data ),
211 |                 ff_hr( &Data ), ff_min( &Data ), 2 * ff_sec( &Data ),
212 |                 ff_attr( &Data ));
213 |         /*  And go to the next node if there is one.
214 |         */
215 |     }while( LLSnodePtr2Next( List ));
216 | }
217 |
218 | void FileOutput( int List ) /* Nearly identical to DirOutput */
219 | {
220 |     DOSFileData Data;
221 |
222 |     if( ! LLSnodePtr2First( List ))
223 |     {
224 |         return; /* list is empty */
225 |     }
226 |
227 |     do
228 |     {
229 |         LLSnodeDataTo( List, &Data );
230 |         printf( "%-14.14s  %10.1ld  "
231 |                 "%4.4d-%2.2d-%2.2d  "
232 |                 "%2.2d:%2.2d:%2.2d  "
233 |                 "%2.2x\n",
234 |                 ff_name( &Data ), ff_size( &Data ),
235 |                 1980 + ff_yr( &Data ), ff_mo( &Data ), ff_day( &Data ),
236 |                 ff_hr( &Data ), ff_min( &Data ), 2 * ff_sec( &Data ),
237 |                 ff_attr( &Data ));
238 |
239 |     }while( LLSnodePtr2Next( List ));
240 | }
241 |
242 | /* === LL_demo.c ========================================================= */
```

```
TEXT STATISTICS

  8980 characters
   242 lines

LEXICAL STATISTICS

    34 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     1 constants [character]
    13 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    CompareResult           52      98     118     152     172
    D1              43      44     147     150     160     166     170
    D2              43      44     147     149     160     165     169
    DOSFileData             58      62      63     149     150     165     166     169     170     189
           220
    Data           189     202     209     210+    211+    212     220     229     234+    235+    236+
           237
    DirComp         43      87     147
    DirOutput       49     136     180
    EXIT_FAILURE            68
    EXIT_SUCCESS           144
    FIND_END       131
    FIND_FIRST              65
    FIND_NEXT      129
    FileComp        44     117     160
    FileOutput              50     141     218
    Info            58      66      73      77      87      99     101     106     117     119     121
           129     131
    LLScreate       62      63
    LLSdelete      137     142
    LLSnodeAdd             101
    LLSnodeAddFrom         121
    LLSnodeDataTo          202     229
    LLSnodeFind             87     117
    LLSnodeInsert           99
    LLSnodeInsertFrom      119
    LLSnodePtr2First        82     116     193     222
    LLSnodePtr2Next        215     239
    List            49      50     180     193     202     215     218     222     229     239
    ListDirs        62      82      87      99     101     136     137
    ListFiles       63     116     117     119     121     141     142
    RetVal         149     152     157     165     168     169     172     177
    Success_        65     129
    _A_ANY          66
    _A_SUBDIR       73
    _A_VOLID       106
    argc            56      65
    argv            56      65
    ff_attr         73     106     212     237
    ff_date        165     166
    ff_day         210     235
    ff_hr          211     236
    ff_min         211     236
    ff_mo          210     235
    ff_name         77     149     150     209     234
    ff_sec         211     236
    ff_size        234
    ff_time        169     170
    ff_yr          210     235
    h       36
    main            56
    printf         205     230
    stdio           36
    strcmp         149
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Here's an example of how to sort a singly-linked list.  I think it
** can be modified to sort a doubly-linked list, but it would get a bit
** more complicated.  Note that this is a recursive method, but the
** recursion depth is limited to be proportional to the base 2 log of
** the length of the list, so it won't "run away" and blow the stack.
**
**  10/21/93 rdg  Fixed bug -- function was okay, but called incorrectly.
*/

#include "snipsort.h"

/* linked list sort -- public domain by Ray Gardner  5/90 */

#include <stdio.h>              /* for NULL definition */
#include <string.h>


/* returns < 0 if *p sorts lower than *q */
int keycmp (list *p, list *q)
{
      return strcmp(p->key, q->key);
}

/* merge 2 lists under dummy head item */
list *lmerge (list *p, list *q)
{
      list *r, head;

      for ( r = &head; p && q; )
      {
            if ( keycmp(p, q) < 0 )
            {
                  r = r->next = p;
                  p = p->next;
            }
            else
            {
                  r = r->next = q;
                  q = q->next;
            }
      }
      r->next = (p ? p : q);
      return head.next;
}

/* split list into 2 parts, sort each recursively, merge */
list *lsort (list *p)
{
      list *q, *r;

      if ( p )
      {
            q = p;
            for ( r = q->next; r && (r = r->next) != NULL; r = r->next )
                  q = q->next;
            r = q->next;
            q->next = NULL;
            if ( r )
                  p = lmerge(lsort(r), lsort(p));
      }
      return p;
}

#if 0       /* Not really main(), but a fill-in-the-blanks framework    */

#ifdef __WATCOMC__
 #pragma off (unreferenced)
#endif

main (void)
{
      list *listp;                /* pointer to start of list */

      /* first build unsorted list, then */

      listp = lsort(listp);                                 /* rdg 10/93 */

      return 0;
}

#endif
```

TEXT STATISTICS

```
  2038 characters
    84 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    NULL            57      60
    __WATCOMC__             69
    h       17      18
    head            30      32      46
    key             24+
    keycmp          22      34
    list            22+     28+     30      50+     52      75
    listp           75      79+
    lmerge          28      62
    lsort           50      62+     79
    main            73
    next            36      37      41      42      45      46      57+     58      59      60
    off             70
    p       22      24      28      32      34      36      37+     45+     50      54      56      62+
            64
    q       22      24      28      32      34      41      42+     45      52      56      57      58+
            59      60
    r       30      32      36+     41+     45      52      57+     59      61      62
    stdio           17
    strcmp          24
    string          18
    unreferenced            70
```

```
/* +++Date last modified: 05-Jul-1997 */

#include    <stdlib.h>
#include    "snipsort.h"


/*
    This is a quicksort routine to be used to sort linked-lists
    by Jon Guthrie.
*/

void    *sortl(void *list, void *(*getnext)(void *),
            void (*setnext)(void *, void *),
            int (*compare)(void *, void *))
{
void    *low_list, *high_list, *current, *pivot, *temp;
int     result;

    /*
        Test for empty list.
    */
    if(NULL == list)
        return(NULL);

    /*
        Find the first element that doesn't have the same value as the first
        element.
    */
    current = list;
    do
    {
        current = getnext(current);
        if(NULL == current)
            return(list);
    }   while(0 == (result = compare(list, current)));

    /*
        My pivot value is the lower of the two.  This insures that the sort
        will always terminate by guaranteeing that there will be at least one
        member of both of the sublists.
    */
    if(result > 0)
        pivot = current;
    else
        pivot = list;

    /* Initialize the sublist pointers */
    low_list = high_list = NULL;

    /*
        Now, separate the items into the two sublists
    */
    current = list;
    while(NULL != current)
    {
        temp = getnext(current);
        if(compare(pivot, current) < 0)
        {
            /* add one to the high list */
            setnext(current, high_list);
            high_list = current;
        }
        else
        {
            /* add one to the low list */
            setnext(current, low_list);
            low_list = current;
        }
        current = temp;
    }

    /*
        And, recursively call the sort for each of the two sublists.
    */
    low_list  = sortl(low_list, getnext, setnext, compare);
    high_list = sortl(high_list, getnext, setnext, compare);

    /*
        Now, I have to put the "high" list after the end of the "low" list.
        To do that, I first have to find the end of the "low" list...
    */
    current = temp = low_list;
    while(1)
    {
        current = getnext(current);
        if(NULL == current)
            break;
        temp = current;
    }

    /*
        Then, I put the "high" list at the end of the low list
    */
    setnext(temp, high_list);
    return(low_list);
}
```

TEXT STATISTICS

2402 characters
96 lines

LEXICAL STATISTICS

12 comments [std-C]
0 comments [C++]
2 preprocessor instructions
0 constants [character]
1 constants [string]
4 constants [numeric]

SYMBOL TABLE

NULL	22	23	33	48	54	86					
compare	14	35	57	75	76						
current	16	29	32+	33	35	43	53	54	56	57	60
61	66	67	69	82	85+	86	88				
getnext	12	32	56	75	76	85					
h	3										
high_list	16	48	60	61	76+	94					
list	12	22	29	34	35	45	53				
low_list	16	48	66	67	75+	82	95				
pivot	16	43	45	57							
result	17	35	42								
setnext	13	60	66	75	76	94					
sortl	12	75	76								
stdlib	3										
temp	16	56	69	82	88	94					

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LL_QUE.h        Generic Queues for fixed size data-items
                    based on the LLS Singly Linked List module.

                    v0.51  94-08-20

                    Note that a LLS version with a 'last node pointer'
                    has some performance advantages on 'putting'.
                    Note that the List's current node pointer always is
                    the same as its first node pointer.
                    The ...Get... 'functions' cannot be used as parameters
                    to other functions.

                    This module has no related .C file.

 _____              This version is Public Domain.
 /_|__|             A.Reitsma, Delft, The Netherlands.
/  | \  ------------------------------------------------------------ */

#ifndef LL_QUE__H
#define LL_QUE__H

#include "lls.h"

#define LLQueCreate(itemsize)   LLScreate( itemsize )
#define LLQueDelete(que)        LLSdelete( que )

#define LLQuePut(que,data)      LLSnodeAppend( que, data )
#define LLQuePutFrom(que,src)   LLSnodeAppendFrom( que, src )

#define LLQueRepair(que)        LLSnodePtr2First( que )
          /* For 'repair' purposes and also to check for an empty Queue */

/* ---- stored data management -------------------------------------------
   return typed data:
*/
#define LLQuePeekInt(que)       LLSnodeInt( que )
#define LLQuePeekLong(que)      LLSnodeLong( que )
#define LLQuePeekPtr(que)       LLSnodePtr( que )
#define LLQuePeekFptr(que)      LLSnodeFptr( que )

#define LLQueGetInt(que)        LLSnodeInt( que ),  LLSnodeDelete( que )
#define LLQueGetLong(que)       LLSnodeLong( que ), LLSnodeDelete( que )
#define LLQueGetPtr(que)        LLSnodePtr( que ),  LLSnodeDelete( que )
#define LLQueGetFptr(que)       LLSnodeFptr( que ), LLSnodeDelete( que )

/* 'return' typeless data.
*/
#define LLQuePeekDataTo(que,dest)     LLSnodeDataTo( que, dest )
#define LLQueGetDataTo(que,dest)      LLSnodeDataTo( que, dest ),        \
                                        (dest ? LLSnodeDelete( que ): 0 )

#endif /* LL_QUE__H */
/* ==== LL_QUE.h  end ================================================ */
```

```
TEXT STATISTICS

  2308 characters
    56 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
    19 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]

SYMBOL TABLE

    LLQueCreate             27
    LLQueDelete             28
    LLQueGetDataTo          52
    LLQueGetFptr            47
    LLQueGetInt             44
    LLQueGetLong            45
    LLQueGetPtr             46
    LLQuePeekDataTo         51
    LLQuePeekFptr           42
    LLQuePeekInt            39
    LLQuePeekLong           40
    LLQuePeekPtr            41
    LLQuePut        30
    LLQuePutFrom            31
    LLQueRepair             33
    LLScreate       27
    LLSdelete       28
    LLSnodeAppend           30
    LLSnodeAppendFrom       31
    LLSnodeDataTo           51      52
    LLSnodeDelete           44      45      46      47      53
    LLSnodeFptr             42      47
    LLSnodeInt              39      44
    LLSnodeLong             40      45
    LLSnodePtr              41      46
    LLSnodePtr2First        33
    LL_QUE__H       22      23
    data            30+
    dest            51+     52+     53
    itemsize        27+
    que             28+     30+     31+     33+     39+     40+     41+     42+     44+     45+     46+
            47+     51+     52+     53
    src             31+
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    LL_STACK.h      Generic Stacks for fixed size data-items
                    based on the LLS Singly Linked List module.

                    v0.50  94-08-12

                    Note that the List's current node pointer always is
                    the same as its first node pointer.
                    This module has no related .C file.

 _____              This version is Public Domain.
/_|__|              A.Reitsma, Delft, The Netherlands.
 | \  ------------------------------------------------------------ */

#ifndef LL_STACK__H
#define LL_STACK__H

#include "lls.h"

#define LLstackCreate(itemsize)        LLScreate( itemsize )
#define LLstackDelete(stack)           LLSdelete( stack )

#define LLstackPush(stack,data)        LLSnodePrepend( stack, data )
#define LLstackPushFrom(stack,source)  LLSnodePrependFrom( stack, source )

#define LLstackRepair(stack)           LLSnodePtr2First( stack )
          /* For 'repair' purposes and also to check for an empty Stack */

/* ---- stored data management -------------------------------------------
   'return' typed data:
*/
#define LLstackPopInt(stack)       LLSnodeInt(stack), LLSnodeDelete(stack)
#define LLstackPopLong(stack)      LLSnodeLong(stack),LLSnodeDelete(stack)
#define LLstackPopPtr(stack)       LLSnodePtr(stack), LLSnodeDelete(stack)
#define LLstackPopFptr(stack)      LLSnodeFptr(stack),LLSnodeDelete(stack)

/* 'return' typeless data.
*/
#define LLstackPopTo(stack,dest)   LLSnodeDataTo(stack,dest),          \
                                       (dest ? LLSnodeDelete(stack) : 0 )

#endif /* LL_STACK__H */
/* ==== LL_STACK.h  end ================================================ */
```

TEXT STATISTICS

```
  1839 characters
    45 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    14 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    LLScreate       22
    LLSdelete       23
    LLSnodeDataTo           41
    LLSnodeDelete           34      35      36      37      42
    LLSnodeFptr             37
    LLSnodeInt              34
    LLSnodeLong             35
    LLSnodePrepend          25
    LLSnodePrependFrom              26
    LLSnodePtr              36
    LLSnodePtr2First        28
    LL_STACK__H             17      18
    LLstackCreate           22
    LLstackDelete           23
    LLstackPopFptr          37
    LLstackPopInt           34
    LLstackPopLong          35
    LLstackPopPtr           36
    LLstackPopTo            41
    LLstackPush             25
    LLstackPushFrom         26
    LLstackRepair           28
    data            25+
    dest            41+     42
    itemsize        22+
    source          26+
    stack           23+     25+     26+     28+     34+     35+     36+     37+     41+     42
```

```
/* +++Date last modified: 05-Jul-1997 */

//
// loctm.h
// Time functions class (a C++ wrapper for ANSI struct tm)
//
// written by David Nugent
//
// This code is public domain. Use for any purpose is unrestricted.
//

# if !defined( _loctm_h )
# define _loctm_h 1
# include <time.h>
# if defined( __MSDOS__ ) || defined( MSDOS )
#  define STRSTREAM_H "strstrea.h"  // 8.3 (sigh)
# else
#  define STRSTREAM_H "strstream.h"
# endif

    // Forward declare for streams interface

class ostream;
class istream;

    // class loc_tm
    // C++ class (mainly I/O) wrapper for ANSI-C struct tm

class loc_tm : public tm
{

  public:

    loc_tm (void) {}                  // Do nothing constructor
    loc_tm (time_t t);     // Set to time, 0 is magic for 'now'
    loc_tm (tm const & t); // Copy constructor

    loc_tm & operator= (tm const & t);            // Assignment
    operator time_t (void) const;        // Conversion operator

    int is_valid (void) const;             // Test for validity

    int compare (loc_tm const & t) const;      // Compare times
    int compare (time_t const tt) const;       // Compare times

    enum f_date     // Date format flags
    {
        d_Year          = (int)0x0001,             // Print year
        d_YearShort     = (int)0x0002, // Print last two digits
        d_PadYear       = (int)0x0004,    // Pad year to 2 or 4
        d_ZeroYear      = (int)0x0008,        // Zero fill year
        d_Month         = (int)0x0010,           // Print month
        d_MonText       = (int)0x0020,   // Print month in text
        d_PadMon        = (int)0x0040, // Pad to 2 (trunc to 3)
        d_ZeroMon       = (int)0x0080,       // Zero fill month
        d_Day           = (int)0x0100,             // Print date
        d_DayOfWeek     = (int)0x0200,     // Print day of week
        d_PadDay        = (int)0x0400,         // Pad date to 2
        d_ZeroDay       = (int)0x0800,         // Zero fill day
        d_DMY           = (int)0x0111, // Print date, mth, year
        d_PadDMY        = (int)0x0444,         // Pad all three
        d_ZeroDMY       = (int)0x0888,   // Zero fill all three
        d_YearFirst     = (int)0x1000,      // Print year first
        d_MonFirst      = (int)0x2000,     // Print month first
        d_SepChar       = (int)0x4000,// Separate fields datech
        d_SepSpace      = (int)0x8000  // Separate fields space
    };
    enum date_f                         // Composite date flags
    {
        d_International = (int)(d_DMY|d_MonText|d_PadMon|
                                d_SepSpace),
        d_IntlShort     = (int)(d_DMY|d_MonText|d_PadMon|
                                d_SepSpace|d_YearShort),
        d_Usa           = (int)(d_DMY|d_MonFirst|d_PadDMY|
                                d_ZeroDMY|d_YearShort|d_SepChar),
        d_English       = (int)(d_DMY|d_YearShort|d_PadDMY|
                                d_ZeroDMY|d_SepChar),
        d_Japanese      = (int)(d_DMY|d_YearFirst|d_PadDMY|
                                d_ZeroDMY|d_YearShort|d_MonFirst|
                                d_SepChar),
        d_Full          = (int)(d_DMY|d_DayOfWeek|d_MonText|
                                d_SepSpace)
    };

    enum f_time  // Time format (nb: time zones not implemented)
    {
        t_Secs          = (int)0x0001,         // Print seconds
        t_ZeroSecs      = (int)0x0002,     // Zero fill seconds
        t_PadSecs       = (int)0x0004,      // Pad seconds to 2
        t_SecsAll       = (int)0x0007,
        t_TimeZone      = (int)0x0008,        // Print timezone
        t_Mins          = (int)0x0010,         // Print minutes
        t_ZeroMins      = (int)0x0020,     // Zero fill minutes
        t_PadMins       = (int)0x0040,      // Pad minutes to 2
        t_MinsAll       = (int)0x0070,
        t_TZNumeric     = (int)0x0080,      // Print numeric TZ
        t_Hour          = (int)0x0100,           // Print hour
        t_ZeroHour      = (int)0x0200,        // Zero fill hour
        t_PadHour       = (int)0x0400,  // Pad hour to 2 digits
        t_HourAll       = (int)0x0700,
```

```
101 |         t_24hour        = (int)0x0800,          // 24hour clock
102 |         t_SepChar       = (int)0x1000, // Separate field timech
103 |         t_SepSpace      = (int)0x2000, // Separate fields space
104 |         t_SepAbbrev     = (int)0x4000,     // Add abbreviations
105 |         t_AmPm          = (int)0x8000       // Add 'am' or 'pm'
106 |     };
107 |     enum time_f  // Composite time flags
108 |     {
109 |         t_International = (int)(t_HourAll|t_MinsAll|t_SecsAll|
110 |                                 t_24hour|t_SepChar),
111 |         t_ShortTime     = (int)(t_HourAll|t_MinsAll|t_24hour|
112 |                                 t_SepChar),
113 |         t_ClockTime     = (int)(t_Hour|t_PadHour|t_MinsAll|
114 |                                 t_AmPm|t_SepChar),
115 |         t_LongTime      = (int)(t_Hour|t_PadHour|t_MinsAll|
116 |                                 t_SecsAll|t_SepAbbrev|t_SepSpace),
117 |         t_Millitary     = (int)(t_HourAll|t_MinsAll|t_24hour)
118 |     };
119 |
120 |
121 |     static char timech;    // Character used for time separator
122 |     static char datech;    // Character used for date separator
123 |     static int datefmt;                  // Default date format
124 |     static int timefmt;                  // Default time format
125 |
126 |                                             // Output methods
127 |     ostream & print (ostream & os, int df =datefmt,
128 |                                    int tf =timefmt) const;
129 |     ostream & printTime (ostream & os, int f =timefmt) const;
130 |     ostream & printDate (ostream & os, int f =datefmt) const;
131 |
132 |                                         // Input/parsing methods
133 |     istream & parseTime (istream & is);         // Unimplemented
134 |     istream & parseDate (istream & is);         // Unimplemented
135 |     istream & parse (istream & is);             // Unimplemented
136 |
137 |   private:
138 |                                               // Time suffix
139 |     void tSfx (ostream & os, int fmt, char ch) const;
140 |                                         // Time field formatters
141 |     void pHour (ostream & os, int fmt) const;
142 |     void pMins (ostream & os, int fmt) const;
143 |     void pSecs (ostream & os, int fmt) const;
144 |                                                 // Date suffix
145 |     void dSfx (ostream & os, int fmt) const;
146 |                                         // Date field formatters
147 |     void pDate (ostream & os, int fmt) const;
148 |     void pMonth (ostream & os, int fmt) const;
149 |     void pYear (ostream & os, int fmt) const;
150 |
151 | };
152 |
153 | inline ostream &                        // Stream output method
154 | operator<< (ostream & os, loc_tm const & t)
155 | {   return t.print(os);     }
156 |
157 | inline int                                        // Equality
158 | operator== (loc_tm const & t1, loc_tm const & t2)
159 | {   return int(t1.compare(t2) == 0);    }
160 |
161 | inline int                                      // Inequality
162 | operator!= (loc_tm const & t1, loc_tm const & t2)
163 | {   return int(t1.compare(t2) != 0);    }
164 |
165 | inline int                                       // Less then
166 | operator<  (loc_tm const & t1, loc_tm const & t2)
167 | {   return int(t1.compare(t2) < 0);     }
168 |
169 | inline int                                    // Less than/equal
170 | operator<= (loc_tm const & t1, loc_tm const & t2)
171 | {   return int(t1.compare(t2) <= 0);    }
172 |
173 | inline int                                      // Greater than
174 | operator>  (loc_tm const & t1, loc_tm const & t2)
175 | {   return int(t1.compare(t2) > 0);     }
176 |
177 | inline int                                  // Greater than/equal
178 | operator>=  (loc_tm const & t1, loc_tm const & t2)
179 | {   return int(t1.compare(t2) >= 0);    }
180 |
181 |
182 |     // class loc_date
183 |     // Date formatter
184 |
185 | class loc_date : public loc_tm
186 | {
187 |
188 |   public:
189 |
190 |     loc_date (loc_tm const & t, int dtfmt =loc_tm::datefmt)
191 |         : loc_tm (t), _fmt(dtfmt)   {}
192 |     loc_date (time_t t, int dtfmt =loc_tm::datefmt)
193 |         : loc_tm (t), _fmt(dtfmt)   {}
194 |     int fmt (int f)                 {   return _fmt = f;    }
195 |     int fmt (void) const            {   return _fmt;        }
196 |
197 |     ostream & print (ostream & os) const
198 |         {   return printDate (os, _fmt);    }
199 |
200 |   private:
```

```
201 |
202 |     int _fmt;
203 |
204 | };
205 |
206 | inline ostream &                // Stream output method - date
207 | operator<< (ostream & os, loc_date const & d)
208 | {   return d.print(os);     }
209 |
210 |
211 |     // class loc_time
212 |     // Time formatter
213 |
214 | class loc_time : public loc_tm
215 | {
216 |
217 |   public:
218 |
219 |     loc_time (loc_tm const & t, int tmfmt =loc_tm::timefmt)
220 |         : loc_tm (t), _fmt(tmfmt)   {}
221 |     loc_time (time_t t, int tmfmt =loc_tm::timefmt)
222 |         : loc_tm (t), _fmt(tmfmt)   {}
223 |     int fmt (int f)                 {   return _fmt = f;    }
224 |     int fmt (void) const            {   return _fmt;        }
225 |
226 |     ostream & print (ostream & os) const
227 |         {   return printTime (os, _fmt);    }
228 |
229 |   private:
230 |
231 |     int _fmt;
232 |
233 | };
234 |
235 | inline ostream &                // Stream output method - time
236 | operator<< (ostream & os, loc_time const & t)
237 | {   return t.print(os);     }
238 |
239 |
240 |
241 | # endif     // _loctm_h
```

```
TEXT STATISTICS

  9146 characters
   241 lines

LEXICAL STATISTICS

     1 comments [std-C]
    86 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     2 constants [string]
    45 constants [numeric]

SYMBOL TABLE

    MSDOS           15
    STRSTREAM_H             16      18
    __MSDOS__       15
    _fmt           191     193     194     195     198     202     220     222     223     224     227
           231
    _loctm_h        12      13
    ch             139
    compare         43      44     159     163     167     171     175     179
    d      207     208
    dSfx           145
    d_DMY           60
    d_DMY|d_DayOfWeek|d_MonText|            81
    d_DMY|d_MonFirst|d_PadDMY|              74
    d_DMY|d_MonText|d_PadMon|       70      72
    d_DMY|d_YearFirst|d_PadDMY|             78
    d_DMY|d_YearShort|d_PadDMY|             76
    d_Day          56
    d_DayOfWeek             57
    d_English       76
    d_Full          81
    d_International         70
    d_IntlShort             72
    d_Japanese              78
    d_MonFirst              64
    d_MonText       53
    d_Month         52
    d_PadDMY        61
    d_PadDay        58
    d_PadMon        54
    d_PadYear       50
    d_SepChar       65      80
    d_SepSpace              66      71      82
    d_SepSpace|d_YearShort          73
    d_Usa          74
    d_Year         48
    d_YearFirst             63
    d_YearShort             49
    d_ZeroDMY       62
    d_ZeroDMY|d_SepChar             77
    d_ZeroDMY|d_YearShort|d_MonFirst|       79
    d_ZeroDMY|d_YearShort|d_SepChar         75
    d_ZeroDay       59
    d_ZeroMon       55
    d_ZeroYear              51
    date_f          68
    datech         122
    datefmt        123     127     130     190     192
    define          13      16      18
    defined         12      15+
    df             127
    dtfmt          190     191     192     193
    endif           19     241
    f      129     130     194+    223+
    f_date          46
    f_time          85
    fmt            139     141     142     143     145     147     148     149     194     195     223
           224
    h       14
    include         14
    is             133     134     135
    is_valid        41
    istream         24     133+    134+    135+
    loc_date       185     190     192     207
    loc_time       214     219     221     236
    loc_tm          29      34      35      36      38      43     154     158+    162+    166+    170+
           174+    178+    185     190+    191     192     193     214     219+    220     221     222
    os             127     129     130     139     141     142     143     145     147     148     149
           154     155     197     198     207     208     226     227     236     237
    ostream         23     127+    129+    130+    139     141     142     143     145     147     148
           149     153     154     197+    206     207     226+    235     236
    pDate          147
    pHour          141
    pMins          142
    pMonth         148
    pSecs          143
    pYear          149
    parse          135
    parseDate      134
    parseTime      133
    print          127     155     197     208     226     237
    printDate      130     198
    printTime      129     227
    t       35      36      38      43     154     155     190     191     192     193     219     220
           221     222     236     237
```

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************************
* LOG.C - Creates a ASCII file with time and date stamps for logging    *
*         hours.                                                        *
*                                                                       *
*   usage: LOG [IN][OUT][CALC]                                          *
*               IN - Creates an opening entry from which a "time spent" *
*                    is calculated.                                     *
*              OUT - Creates a closing entry and calculates             *
*                    "time spent" for that entry.                       *
*             CALC - Calculates total overall "time spent" for the      *
*                    entire log.                                        *
*                                                                       *
*   NOTES:  I used seconds to do all the calculations. The other        *
*           time/date entries are for human readability. Some           *
*           enhancements can be done to this program.                   *
*           i.e. Wage/Pay calculation, closing the log after a CALC     *
*           to insure log is not reused, tracking hours for individual  *
*           people, tracking hours for individual projects, etc.        *
*                                                                       *
*  Public domain by Robert Sprawls.                                     *
*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <time.h>

/* Define time file constants   */

#define HOUR        3600        /* Number of seconds in an hour.        */
#define MINS        60          /* Number of seconds in a minute        */
#define IN_ENTRY    40          /* Size of an IN entry                  */
#define SEC_OFF     4           /* Offset of seconds in an IN entry     */
#define HOUR_OFF    64          /* Offset of seconds in an "time spent" */
                                /* calculated entry.                    */

/* Define values returned to DOS */

#define OK          0   /* Executed normally, nothing will be echoed */
#define OPENLOG     1   /* Attempted to log in while open entry in log */
#define CLOSEDLOG   2   /* Attempted to log out while closed entry in log */
#define FILE_ERROR  3   /* File access error. Just in case. */
#define SEEK_ERROR  4   /* File positioning error */
#define NOPARMS     5   /* No parameters supplied to program */
#define INVALID     6   /* Invalid parameters */

void    usage( void );
long    get_in_entry( FILE * );
void    fastforw( FILE * );     /* Opposite of rewind();        */
void    quit( int );

char    strbuf[ IN_ENTRY + 1 ];
FILE    *wrklog;

int main( int argc, char *argv[] )
{
    char    outline[ IN_ENTRY * 2 + 1 ];
    long    in_entry_time = 0, total_seconds = 0;
    time_t  current_time;
    div_t   hdiv, mdiv;

    if( argc < 2 )
    {
        usage();
        quit( NOPARMS );
    }

    /* Open log. Can be any directory.  */
    if(( wrklog = fopen( "WORK.LOG", "a+" )) == NULL )
        quit( FILE_ERROR );

    strupr( argv[ 1 ] );

    time( &current_time );

    /* Create an opening IN entry.  */
    if( strcmp( "IN", argv[ 1 ] ) == 0 )
    {
        /* Make sure there isn't a open entry already.  */
        if( get_in_entry( wrklog ) )
            quit( OPENLOG );

        /* Stamp it.    */
        fprintf( wrklog, "%3s %ld %s", argv[ 1 ], current_time,
            ctime( &current_time ));
    }
    /* Create a closing OUT entry.  */
    else if( strcmp( "OUT", argv[ 1 ] ) == 0 )
    {
        /* Make sure there is a previous IN entry.  */
        if( ( in_entry_time = get_in_entry( wrklog )) == 0 )
            quit( CLOSEDLOG );

        total_seconds = current_time - in_entry_time;
        sprintf( outline, "%3s %ld %s", argv[ 1 ], current_time,
            ctime( &current_time ));
```

```
101 |         /* Cut off the newline character that's normally put on.    */
102 |         outline[ strlen( outline ) - 1 ] = '\0';
103 |         hdiv = div( total_seconds, HOUR );
104 |         mdiv = div( hdiv.rem, MINS );
105 |
106 |         sprintf( strbuf, "     Time Spent: %02d:%02d:%02d/%ld\n\n",
107 |             hdiv.quot, mdiv.quot, mdiv.rem, total_seconds );
108 |         strcat( outline, strbuf );
109 |         fprintf( wrklog, outline );
110 |     }
111 |     /* Calculate the overall "time spent"   */
112 |     else if( strcmp( "CALC", argv[ 1 ] ) == 0 )
113 |     {
114 |         rewind( wrklog );
115 |         while( !feof( wrklog ) )
116 |         {
117 |             /* This is to eliminate garbage or residual entries.    */
118 |             outline[ 0 ] = '\0';
119 |
120 |             fgets( outline, 80, wrklog );
121 |             if( strstr( outline, "OUT" ) != NULL )
122 |             {
123 |                 total_seconds += atol( &outline[ HOUR_OFF ] );
124 |             }
125 |
126 |         }
127 |
128 |         /* goto to end of file and stamp total hours    */
129 |         fastforw( wrklog );
130 |         if( total_seconds )
131 |         {
132 |             hdiv = div( total_seconds, HOUR );
133 |             mdiv = div( hdiv.rem, MINS );
134 |             fprintf( wrklog, "\t\t\t\t\t\t\t\t\t\t  "
135 |                 "Total Hours: %02d:%02d:%02d/%ld\n",
136 |                 hdiv.quot, mdiv.quot, mdiv.rem, total_seconds );
137 |         }
138 |     }
139 |     else
140 |     {
141 |         usage();
142 |         quit( INVALID );
143 |     }
144 |
145 |     quit( OK );
146 |     return 0;
147 | }
148 |
149 | void usage( void )
150 | {
151 |     printf( "\nusage: LOG [IN][OUT][CALC]\n");
152 | }
153 |
154 | /*************************************************************
155 | * get_in_entry - gets a previous IN entry.                   *
156 | *                                                            *
157 | *  Params: FILES *fp - file pointer.                         *
158 | * Returns: The entry's seconds if successful, else 0         *
159 | *                                                            *
160 | * NOTES: If fseek fails for any reason, function does not    *
161 | *        return. Instead, quit is call with the error code.  *
162 | *************************************************************/
163 |
164 | long get_in_entry( FILE *fp )
165 | {
166 |
167 |     if( fseek( fp, -IN_ENTRY, SEEK_END ) != 0 )
168 |         quit( SEEK_ERROR );
169 |
170 |     fread( strbuf, 1, IN_ENTRY, fp );
171 |     fastforw( fp );
172 |
173 |     if( strstr( strbuf, "IN" ) == NULL )
174 |         return( 0 );
175 |     else
176 |     {
177 |         return( atol( &strbuf[ SEC_OFF ]));
178 |     }
179 | }
180 |
181 | /*************************************************************
182 | * quit() - Program exit function. Reports of any outstanding *
183 | *          errors.                                           *
184 | *                                                            *
185 | *  Params:  errcode - Error code as defined in beginning.    *
186 | * Returns:  nothing.                                         *
187 | *************************************************************/
188 |
189 | void quit( int errcode )
190 | {
191 |     char *errmsg[] =
192 |     {
193 |         "",
194 |         "Log has an open entry.",
195 |         "No corresponding IN entry.",
196 |         "File open error.",
197 |         "Seek error",
198 |         "No parameters specified.",
199 |         "Invalid Parameters."
200 |     };
```

```
 201 |
 202 |     printf( "\n%s\n", errmsg[ errcode ] );
 203 |
 204 |     fclose( wrklog );
 205 |     exit( errcode );
 206 | }
 207 |
 208 | /*************************************************************
 209 | * fastforw() - Puts file pointer to end of file.             *
 210 | *                                                            *
 211 | *  Params: fp - File pointer.                                *
 212 | * Returns: nothing.                                          *
 213 | *************************************************************/
 214 |
 215 | void fastforw( FILE *fp )
 216 | {
 217 |     fseek( fp, 0, SEEK_END );
 218 | }
```

TEXT STATISTICS

```
  7876 characters
   218 lines
```

LEXICAL STATISTICS

```
    31 comments [std-C]
     0 comments [C++]
    17 preprocessor instructions
     2 constants [character]
    21 constants [string]
    36 constants [numeric]
```

SYMBOL TABLE

```
    CLOSEDLOG       44      95
    FILE            51      52      56     164     215
    FILE_ERROR              45      73
    HOUR            33     103     132
    HOUR_OFF        37     123
    INVALID         48     142
    IN_ENTRY        35      55      60     167     170
    MINS            34     104     133
    NOPARMS         47      68
    NULL            72     121     173
    OK              42     145
    OPENLOG         43      84
    SEC_OFF         36     177
    SEEK_END       167     217
    SEEK_ERROR              46     168
    argc            58      65
    argv            58      75      80      87      91      98     112
    atol           123     177
    ctime           88      99
    current_time            62      77      87      88      97      98      99
    div            103     104     132     133
    div_t           63
    errcode        189     202     205
    errmsg         191     202
    exit           205
    fastforw        52     129     171     215
    fclose         204
    feof           115
    fgets          120
    fopen           72
    fp             164     167     170     171     215     217
    fprintf         87     109     134
    fread          170
    fseek          167     217
    get_in_entry            51      83      94     164
    h       25      26      27      28      29
    hdiv            63     103     104     107     132     133     136
    in_entry_time           61      94      97
    main            58
    mdiv            63     104     107+    133     136+
    outline         60      98     102+    108     109     118     120     121     123
    printf         151     202
    quit            53      68      73      84      95     142     145     168     189
    quot           107+    136+
    rem            104     107     133     136
    rewind         114
    sprintf         98     106
    stddef          27
    stdio           26
    stdlib          25
    strbuf          55     106     108     170     173     177
    strcat         108
    strcmp          80      91     112
    string          28
    strlen         102
    strstr         121     173
    strupr          75
    time            29      77
    time_t          62
    total_seconds           61      97     103     107     123     130     132     136
    usage           50      67     141     149
    wrklog          56      72      83      87      94     109     114     115     120     129     134
           204
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  L O G S C A L E
**
**  Logarithmically scale from long integer to long integer. The
**  function domain (input) & range (output) both start from 0
**  and the range MUST be smaller than the domain.
**
**  This scaling function was especially designed to colour fractals.
**  In order to preserve detail & to ensure that no colours are wasted,
**  the function is almost linear with a slope near 1 for early values.
**
**  Written by M. Stapleton of Graphic Bits
**
**  Public domain
**
**  May 18 1997
*/

#include "snipmath.h"


/* "Shifted" logarithm = log(u+v) - log(v) */

#define shlog(u,v) (log(((double)(u) + (double)(v)) / (double)(v)))

static double alpha;            /* Scaling parameter */

/*
**  Calculate scaling parameter (alpha) using Newton's method,
**  where rmax = alpha * shlog(dmax, alpha)
**  Boolean return: 0 on failure
*/

int initlogscale(long dmax, long rmax)
{
      double dm = (double) dmax;
      double rm = (double) rmax;
      double x = 1.0, dx, y, yy, t;

      if (dm <= rm)
            return 0;

      do
      {
            t = shlog(dm, x);
            y = x * t - rm;
            yy = t - dm / (x + dm);
            dx = y / yy;
            x -= dx;
            /* printf("Alpha = %f\n", x); */
      } while (abs(dx) > 1E-4);

      alpha = x;

      return 1;
}

/*
**  Logarithmically scale d, offset by alpha
*/

long logscale(long d)
{
      double r = alpha * shlog(d, alpha);

      return (long) floor(0.5 + r);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

/*
**  Default domain & range maxima.
*/

#define DMAX 1000
#define RMAX 255

/*
**  Test logscale
*/

int main(int argc, char *argv[])
{
      long d, dmax = 0, r0, r1, rmax = 0;

      if (argc == 2 && *argv[1] == '?')
      {
            printf("Test logscale()\n Usage:\n");
            printf("%s [domain] [range]\n", argv[0]);
            return EXIT_FAILURE;
      }

      /*
```

```
101 |       **  Get domain & range maxima
102 |       */
103 |
104 |       if (argc > 1)
105 |             dmax = atol(argv[1]);
106 |       if (dmax == 0)
107 |             dmax = DMAX;
108 |
109 |       if (argc > 2)
110 |             rmax = atol(argv[2]);
111 |       if (rmax == 0)
112 |             rmax = RMAX;
113 |
114 |       if (rmax >= dmax)
115 |       {
116 |             printf("Warning: range must be smaller than domain!\n");
117 |
118 |             /*
119 |             **  A real program we would exit here, but we go on to test
120 |             **  initlogscale's error handling...
121 |             */
122 |       }
123 |
124 |       /*
125 |       ** Initialise scaling parameter
126 |       */
127 |
128 |       if (!initlogscale(dmax, rmax))
129 |       {
130 |             printf("Error: cannot initialise logscale!\n");
131 |             return EXIT_FAILURE;
132 |       }
133 |
134 |       /*
135 |       **  Calculate function for every domain value.
136 |       ** Only print where the range value changes.
137 |       */
138 |
139 |       for (r0 = -1, d = 0; d < dmax; d++)
140 |       {
141 |             r1 = logscale(d);
142 |             if (r1 != r0)
143 |             {
144 |                   printf("%ld -> %ld\n", d, r1);
145 |                   r0 = r1;
146 |             }
147 |       }
148 |       printf("%ld -> %ld\n", d, r1);
149 |
150 |       return EXIT_SUCCESS;
151 | }
152 |
153 | #endif /* TEST */
```

```
TEXT STATISTICS

  3312 characters
   153 lines

LEXICAL STATISTICS

    15 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     1 constants [character]
     7 constants [string]
    20 constants [numeric]

SYMBOL TABLE

    DMAX            82     107
    EXIT_FAILURE     97     131
    EXIT_SUCCESS    150
    RMAX            83     112
    TEST            73
    abs             53
    alpha           28      55      66+
    argc            89      93     104     109
    argv            89      93      96     105     110
    atol           105     110
    d       64      66      91     139+    141     144     148
    dm              38      42      47      49+
    dmax            36      38      91     105     106     107     114     128     139
    dx              40      50      51      53
    floor           68
    h       75      76
    initlogscale     36     128
    log             26
    logscale        64     141
    main            89
    printf          95      96     116     130     144     148
    r       66      68
    r0              91     139     142     145
    r1              91     141     142     144     145     148
    rm              39      42      48
    rmax            36      39      91     110     111     112     114     128
    shlog           26      47      66
    stdio           75
    stdlib          76
    t       40      47      48      49
    u       26+
    v       26+
    x       40      47      48      49      51      55
    y       40      48      50
    yy              40      49      50
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  LOOKUP.C
**
**  This is the lookup function for reading a binary file index to an ASCII
**  file formatted as follows:
**
**  Mark Corgan
**  550 Foothill Rd.
**  Gardnerville, NV 89410
**  (702) 265-2388
**  .
**  Hello World
**  123 Anywhere St.
**  Anytown, CA 12345
**  (123) 456-7890
**  .
**  etc...
**
**  The period is what  LOOKUP.C looks for to indicate the end
**  of record. Of course, you could have any format you like, so long as the
**  first line is the information you are looking for. Also, there is no
**  limit to the number of lines of infomation after the first line and
**  before the period as fputs() continues until the period. Enjoy!
**
**  by Mark Corgan, 09-May-1993, and donated to the public domain
*/

#include <stdio.h>
#include "errors.h"
#include "indxlook.h"

long bsearch_(FILE *ifp, long first, long last, char *target);

int main(int argc, char *argv[])
{
      FILE *afp, *ifp;
      char line[MAX_LINE];
      INDEX header;
      long pos;

      if (argc != 3)
      {
            puts("Usage: LOOKUP text_file_name index_file_name\n");
            puts("Note: The text file must consist of a number of records "
                 "separated by lines");
            puts("      containing a single period (\".\")");
            return EXIT_FAILURE;
      }
      afp = cant(argv[1], "r");
      ifp = cant(argv[2], "rb");
      if (fread((char *) &header, sizeof(INDEX), 1, ifp) == 0)
            fprintf(stderr, "Can't read header of \"%s\"\n", argv[2]);
      else  while(printf("Name? "), fgets(line, sizeof(line), stdin))
      {
            if ((pos = bsearch_(ifp, 1L, header.pos, line)) == -1)
                  printf("Couldn't find: %s\n", line);
            else if (fseek(afp, pos, 0) != 0)
                  fprintf(stderr, "Can't read record at position %ld\n", pos);
            else while(fgets(line,sizeof(line),afp) && strcmp(line,END_REC)!=0)
                  fputs(line, stdout);
      }
      fclose(ifp);
      fclose(afp);

      exit(0);
}

long bsearch_(FILE *ifp, long first, long last, char *target)
{
      long pos, mid =(first + last) / 2;
      INDEX next;
      int cmp;

      if (mid < first || fseek(ifp, mid * sizeof(INDEX), 0) != 0 ||
          fread((char *) &next, sizeof(INDEX), 1, ifp) == 0)
      {
            pos = -1;
      }
      else  pos = ((cmp = strncmp(target, next.key, MAX_KEY)) == 0) ?
                  next.pos : ((cmp < 0) ? bsearch_(ifp, first, mid - 1, target)
                                        : bsearch_(ifp, mid + 1, last, target));
      return pos;
}
```

```
TEXT STATISTICS

  2700 characters
    85 lines

LEXICAL STATISTICS

     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    12 constants [string]
    22 constants [numeric]

SYMBOL TABLE

    END_REC         61
    EXIT_FAILURE            49
    FILE            34      38      70
    INDEX           40      53      73      76      77
    MAX_KEY         81
    MAX_LINE        39
    afp             38      51      59      61      65
    argc            36      43
    argv            36      51      52      54
    bsearch_        34      57      70      82      83
    cant            51      52
    cmp             74      81      82
    exit            67
    fclose          64      65
    fgets           55      61
    first           34      70      72      76      82
    fprintf         54      60
    fputs           62
    fread           53      77
    fseek           59      76
    h       30
    header          40      53      57
    ifp             34      38      52      53      57      64      70      76      77      82      83
    key             81
    last            34      70      72      83
    line            39      55+     57      58      61+     62
    main            36
    mid             72      76+     82      83
    next            73      77      81      82
    pos             41      57+     59      60      72      79      81      82      84
    printf          55      58
    puts            45      46      48
    stderr          54      60
    stdin           55
    stdio           30
    stdout          62
    strcmp          61
    strncmp         81
    target          34      70      81      82      83
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** LSD - A simple directory lister
  5 | ** A public domain C demo program by Bob Stout
  6 | */
  7 |
  8 | #include <stdio.h>
  9 | #include <stdlib.h>
 10 | #include <string.h>
 11 | #include <ctype.h>
 12 | #include "dirport.h"
 13 | #include "sniptype.h"
 14 |
 15 | char *sprintfc(char *, long);
 16 | char *capitalize(const char *);
 17 | int   one_column(char *, char *, long, unsigned, DOSFileDate, DOSFileTime);
 18 | int   five_column(char *, char *);
 19 |
 20 | /*
 21 | **  DOS DIR improved work-alike w/ improved formatting & attribute display
 22 | **
 23 | **  supports /W switch
 24 | */
 25 |
 26 | main(int argc, char *argv[])
 27 | {
 28 |       int i, files = 0, dirs = 0, argptr = 0, errflag = False_, cols, drive;
 29 |       long siz_tot = 0L;
 30 |       char *p, *fname, *ext, name[13], buf[67], numbuf[12];
 31 |       DOSFileData ff;
 32 | #ifndef __ZTC__
 33 |       struct diskfree_t df;
 34 | #endif
 35 |       int (*display)() = one_column;
 36 |
 37 |       strcpy(buf, fname = "*.*");
 38 |       if(argc != 1)   for (i = 1; i < argc; ++i)
 39 |       {
 40 |             if ('/' == argv[i][0])
 41 |             {
 42 |                   if ('W' == toupper(argv[i][1]))
 43 |                         display = five_column;
 44 |                   else
 45 |                   {
 46 |                         puts("\aUsage: LSD [/W] [file]");
 47 |                         errflag = True_;
 48 |                   }
 49 |             }
 50 |             else if (!argptr)
 51 |                   argptr = i;
 52 |       }
 53 |       if (argptr)
 54 |       {
 55 |             fname = argv[argptr];
 56 |             strcpy(buf, fname);
 57 |             if ('\\' == LAST_CHAR(buf) || ':' == LAST_CHAR(buf))
 58 |                   strcat(buf, "*.*");
 59 |             else
 60 |             {
 61 |                   if (Success_ == FIND_FIRST(buf, _A_SUBDIR, &ff))
 62 |                   {
 63 |                         if (ff_attr(&ff) & _A_SUBDIR && '.' != *ff_name(&ff))
 64 |                         {
 65 |                               if ('\\' != LAST_CHAR(buf))
 66 |                                     strcat(buf, "\\");
 67 |                               strcat(buf, "*.*");
 68 |                         }
 69 |                   }
 70 |                   else  errflag = True_;
 71 |             }
 72 |       }
 73 |       if (':' == buf[1])
 74 |             drive = toupper(*buf) - '@';
 75 |       else  drive = 0;
 76 |       if (!errflag && !(FIND_FIRST(buf, _A_ANY, &ff))) do
 77 |       {
 78 |             siz_tot += ff_size(&ff);
 79 |             if (ff_attr(&ff) & _A_SUBDIR)
 80 |                   ++dirs;
 81 |             else  ++files;
 82 |             strcpy(name, ff_name(&ff));
 83 |             if (NULL != (p = strchr(name, '.')) && p != name)
 84 |             {
 85 |                   *p  = '\0';
 86 |                   ext = ++p;
 87 |             }
 88 |             else  ext = "";
 89 |             cols = (*display)(name, ext, ff_size(&ff),
 90 |                   ff_attr(&ff), ff_date(&ff), ff_time(&ff));
 91 |       } while (Success_ == FIND_NEXT(&ff));
 92 |       else
 93 |       {
 94 |             fprintf(stderr, "Cannot do directory on '%s'\n", fname);
 95 |             exit(-1);
 96 |       }
 97 |       FIND_END(&ff);
 98 |       if (cols)
 99 |             fputc('\n', stdout);
100 |       sprintfc(numbuf,siz_tot);
```

```
101 |       printf("\n%3d Files totalling %s bytes\n", files, numbuf);
102 |       printf("%3d Director%s\n", dirs, (1 == dirs) ? "y" : "ies");
103 | #ifndef __ZTC__
104 |       _dos_getdiskfree(drive, &df);
105 |       sprintfc(numbuf, (long)df.avail_clusters * df.sectors_per_cluster *
106 |             df.bytes_per_sector);
107 | #else /* if ZTC */
108 |       sprintfc(numbuf, dos_getdiskfreespace(drive));
109 | #endif
110 |       printf("%s bytes free\n", numbuf);
111 |       return 0;
112 | }
113 |
114 | /*
115 | **  The single column directory entry display function
116 | */
117 |
118 | int one_column(char        *name,
119 |                char        *ext,
120 |                long         size,
121 |                unsigned     attribs,
122 |                DOSFileDate  date,
123 |                DOSFileTime  time)
124 | {
125 |       register int i, mask;
126 |       static char *atr = "RHSVDA", szbuf[12];
127 |
128 |       sprintfc(szbuf, size);
129 |       printf("%-8s %-3s %12s  ", capitalize(name), capitalize(ext), szbuf);
130 |
131 |       for (i = 0, mask = 1; i < 6; ++i, mask <<= 1)
132 |             fputc((attribs & mask) ? atr[i] : '.', stdout);
133 |
134 |       printf("%4d-%02d-%02d%4d:%02d:%02d\n",
135 |             date.month,
136 |             date.day,
137 |             (date.year + 80) % 100,
138 |             time.hours,
139 |             time.mins,
140 |             time.tsecs * 2);
141 |
142 |       return 0;
143 | }
144 |
145 | /*
146 | **  The five column directory entry display function
147 | */
148 |
149 | int five_column(char *name, char *ext)
150 | {
151 |       static int cols = 0;
152 |
153 |       printf("%-8s %-3s%s", capitalize(name), capitalize(ext),
154 |             (5 > ++cols) ? "    " : "");
155 |       if (0 == (cols %= 5))
156 |             putchar('\n');
157 |       return (cols);
158 | }
159 |
160 | /*
161 | **  Display a long int using commas as thousands separators
162 | */
163 |
164 | char *sprintfc(char *string, long num)
165 | {
166 |       if (num > 999999L)
167 |       {
168 |             sprintf(string, "%d,%03d,%03d",
169 |                   (int)(num / 1000000L),
170 |                   (int)((num % 1000000L) / 1000L),
171 |                   (int)(num % 1000L));
172 |       }
173 |       else
174 |       {
175 |             if (num > 999L)
176 |             {
177 |                   sprintf(string, "%d,%03d",
178 |                         (int)(num / 1000L),
179 |                         (int)(num % 1000L));
180 |             }
181 |             else  sprintf(string, "%d", (int)num);
182 |       }
183 |       return string;
184 | }
185 |
186 | /*
187 | **  Capitalize a name or extension in place
188 | */
189 |
190 | char *capitalize(const char *string)
191 | {
192 |       int flag = 0;
193 |       char *ptr = (char *)string;
194 |
195 |       do
196 |       {
197 |             if (isalpha(*ptr))
198 |             {
199 |                   if (flag)
200 |                         *ptr = (char)tolower(*ptr);
```

```
 201 |                   else
 202 |                   {
 203 |                         flag = 1;
 204 |                         *ptr = (char)toupper(*ptr);
 205 |                   }
 206 |             }
 207 |             else  flag = 0;
 208 |       } while (*++ptr);
 209 |       return (char *)string;
 210 | }
```

TEXT STATISTICS

```
  5796 characters
   210 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
    13 constants [character]
    23 constants [string]
    40 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             31
    DOSFileDate             17     122
    DOSFileTime             17     123
    FIND_END        97
    FIND_FIRST              61      76
    FIND_NEXT       91
    False_          28
    LAST_CHAR       57+     65
    NULL            83
    Success_        61      91
    True_           47      70
    _A_ANY          76
    _A_SUBDIR       61      63      79
    __ZTC__         32     103
    _dos_getdiskfree       104
    argc            26      38+
    argptr          28      50      51      53      55
    argv            26      40      42      55
    atr            126     132
    attribs        121     132
    avail_clusters         105
    buf             30      37      56      57+     58      61      65      66      67      73      74
          76
    bytes_per_sector       106
    capitalize              16     129+    153+    190
    cols            28      89      98     151     154     155     157
    ctype           11
    date           122     135     136     137
    day            136
    df              33     104     105+    106
    dirs            28      80     102+
    diskfree_t              33
    display         35      43      89
    dos_getdiskfreespace           108
    drive           28      74      75     104     108
    errflag         28      47      70      76
    exit            95
    ext             30      86      88      89     119     129     149     153
    ff              31      61      63+     76      78      79      82      89      90+     91      97
    ff_attr         63      79      90
    ff_date         90
    ff_name         63      82
    ff_size         78      89
    ff_time         90
    files           28      81     101
    five_column             18      43     149
    flag           192     199     203     207
    fname           30      37      55      56      94
    fprintf         94
    fputc           99     132
    h        8       9      10      11
    hours          138
    i       28      38+     40      42      51     125     131+    132
    isalpha        197
    main            26
    mask           125     131+    132
    mins           139
    month          135
    name            30      82      83+     89     118     129     149     153
    num            164     166     169     170     171     175     178     179     181
    numbuf          30     100     101     105     108     110
    one_column              17      35     118
    p       30      83+     85      86
    printf         101     102     110     129     134     153
    ptr            193     197     200+    204+    208
    putchar        156
    puts            46
    sectors_per_cluster            105
    siz_tot         29      78     100
    size           120     128
    sprintf        168     177     181
    sprintfc        15     100     105     108     128     164
```

```
stderr          94
stdio            8
stdlib           9
stdout          99     132
strcat          58      66      67
strchr          83
strcpy          37      56      82
string          10     164     168     177     181     183     190     193     209
szbuf          126     128     129
time           123     138     139     140
tolower        200
toupper         42      74     204
tsecs          140
year           137
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  LTOA.C -- routine and example program to convert a long int to
**  the specified numeric base, from 2 to 36.
**
**  Written by Thad Smith III, Boulder, CO. USA  9/06/91
**  and contributed to the Public Domain.
**
**  Parameters: 1 - number to be converted
**              2 - buffer in which to build the converted string
**              3 - number base to use for conversion (2-36)
**
**  Returns:  A character pointer to the converted string if
**            successful, a NULL pointer if the number base specified
**            is out of range.
*/

#include <stdlib.h>
#include "numcnvrt.h"

#if defined(__ZTC__) && !defined(__SC__)

char *ltoa(
      long val,                                 /* value to be converted */
      char *buf,                                /* output string         */
      int base)                                 /* conversion base       */
{
      ldiv_t r;                                 /* result of val / base  */

      if (base > 36 || base < 2)          /* no conversion if wrong base */
      {
            *buf = '\0';
            return buf;
      }
      if (val < 0)
            *buf++ = '-';
      r = ldiv (labs(val), base);

      /* output digits of val/base first */

      if (r.quot > 0)
            buf = ltoa ( r.quot, buf, base);

      /* output last digit */

      *buf++ = "0123456789abcdefghijklmnopqrstuvwxyz"[(int)r.rem];
      *buf   = '\0';
      return buf;
}

#endif

#ifdef TEST

#include <stdio.h>

main()
{
      char buf[100], line[100], *tail;
      long v;
      int inbase, outbase;

      for (;;)
      {
            printf ("inbase, value, outbase (or hit Ctrl-C to exit)? ");
            fgets (line, sizeof line, stdin);
            sscanf (line, " %d%*[, ]%[^, ]%*[, ]%d", &inbase, buf, &outbase);
            if (inbase == 0)
                  break;                        /* exit if first number 0 */
            v = strtol (buf, &tail, inbase);
            printf("%s(%d) = %ld(10)\n", buf, inbase, v);
            ltoa ( v, buf, outbase);
            printf("ltoa(%ld, buf, %d) = %s\n\n", v, outbase, buf);
            printf ("=%ld (10) = %s (%d).\n", v, buf, outbase);
      };
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2336 characters
    80 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     3 constants [character]
     7 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    TEST            54
    __SC__          22
    __ZTC__         22
    base            27      31+     38      43
    buf             26      33      34      37      43+     47      48      49      60      68      71
            72      73      74      75
    defined         22+
    fgets           67
    h       19      56
    inbase          62      68      69      71      72
    labs            38
    ldiv            38
    ldiv_t          29
    line            60      67+     68
    ltoa            24      43      73
    main            58
    outbase         62      68      73      74      75
    printf          66      72      74      75
    quot            42      43
    r       29      38      42      43      47
    rem             47
    sscanf          68
    stdin           67
    stdio           56
    stdlib          19
    strtol          71
    tail            60      71
    v       61      71      72      73      74      75
    val             25      36      38
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  LTOSTR.C - An improved, safer, ltoa()
**
**  On call:
**  num       =   number to convert
**  string    =   buffer for output
**  max_chars =   maximum size of buffer
**  base      =   number base for conversion.
**
**  Return value:
**  if illegal base
**    NULL
**  beginning of converted number.
**
**  notes: if number is too large in magnitude to fit in the buffer,
**  the MOST significant digits will be truncated.  If the number is
**  negative, a leading '-' will be placed in the buffer even if this
**  causes the most significant digit to be truncated.
**  The number is right justified in the buffer and the location of the
**  first character in the number is returned so:
**    If you want right justification, use the original string.
**    If you want left justification, use the returned string.
**  If the number doesn't fill the buffer:
**       leading characters will be filled with spaces.
**
**  public domain by Jerry Coffin
*/

#include <stdio.h>
#include <string.h>
#include "numcnvrt.h"

char *ltostr(long num, char *string, size_t max_chars, unsigned base)
{
      char remainder;
      int sign = 0;   /* number of digits occupied by the sign. */

      if (base < 2 || base > 36)
            return NULL;

      if (num < 0)
      {
            sign = 1;
            num = -num;
      }

      string[--max_chars] = '\0';

      for (max_chars--; max_chars > sign && num!=0; max_chars --)
      {
            remainder = (char) (num % base);
            if ( remainder < 9 )
                  string[max_chars] = remainder + '0';
            else  string[max_chars] = remainder - 10 + 'A';
            num /= base;
      }

      if (sign)
            string[--max_chars] = '-';

      if ( max_chars > 0 )
            memset(string, ' ', max_chars+1);

      return string + max_chars;
}

#ifdef TEST

#include <stdlib.h>

#ifdef __WATCOMC__
 #pragma off (unreferenced);
#endif

#ifdef __TURBOC__
 #pragma argsused
#endif

#define SIZE 50

int main(int argc, char *argv[])
{
      char buffer[SIZE];

      long number   = atoi(argv[1]);
      unsigned base = atoi(argv[2]);

      printf("%ld in base %u is \"%s\"\n", number, base,
            ltostr(number, buffer, SIZE, base));

      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2398 characters
    96 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     5 constants [character]
     2 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

NULL		41							
SIZE		81	85	91					
TEST		69							
__TURBOC__			77						
__WATCOMC__			73						
argc		83							
argsused		78							
argv		83	87	88					
atoi		87	88						
base		35	40+	53	57	88	90	91	
buffer		85	91						
h	31	32	71						
ltostr		35	91						
main		83							
max_chars		35	49	51+	55	56	61	63	64	66
memset		64							
num		35	43	46+	51	53	57		
number		87	90	91					
off		74							
printf		90							
remainder		37	53	54	55	56			
sign		38	45	51	60				
size_t		35							
stdio		31							
stdlib		71							
string		32	35	49	55	56	61	64	66
unreferenced			74						

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Originally published as part of the MicroFirm Function Library
**
**  Copyright 1987-88, Robert B.Stout
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Makes all whitespace single spaces. Passed a string, lv1ws()
**  converts all multiple whitespace characters to single spaces.
*/

#include <ctype.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

void lv1ws(char *str)
{
      char *ibuf, *obuf;
      int i, cnt;

      if (str)
      {
            ibuf = obuf = str;
            i = cnt = 0;

            while(*ibuf)
            {
                  if(isspace(*ibuf) && cnt)
                        ibuf++;
                  else
                  {
                        if (!isspace(*ibuf))
                              cnt = 0;
                        else
                        {
                              *ibuf = ' ';
                              cnt = 1;
                        }
                        obuf[i++] = *ibuf++;
                  }
            }
            obuf[i] = '\0';
      }
}

#if defined(__cplusplus) && __cplusplus
 }
#endif
```

TEXT STATISTICS

```
  1370 characters
    55 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     2 constants [character]
     2 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    __cplusplus             19+     53+
    cnt             26      31      35      40      44
    ctype           16
    defined         19      53
    h       16
    i       26      31      46      49
    ibuf            25      30      33      35      36      39      43      46
    isspace         35      39
    lv1ws           23
    obuf            25      30      46      49
    str             23      28      30
```

```
/* +++Date last modified: 05-Jul-1997 */

/**********************************************************
    lzhuf.c
    written by Haruyasu Yoshizaki 1988/11/20
    some minor changes 1989/04/06
    comments translated by Haruhiko Okumura 1989/04/07
    getbit and getbyte modified 1990/03/23 by Paul Edwards
      so that they would work on machines where integers are
      not necessarily 16 bits (although ANSI guarantees a
      minimum of 16).  This program has compiled and run with
      no errors under Turbo C 2.0, Power C, and SAS/C 4.5
      (running on an IBM mainframe under MVS/XA 2.2).  Could
      people please use YYYY/MM/DD date format so that everyone
      in the world can know what format the date is in?
    external storage of filesize changed 1990/04/18 by Paul Edwards to
      Intel's "little endian" rather than a machine-dependant style so
      that files produced on one machine with lzhuf can be decoded on
      any other.  "little endian" style was chosen since lzhuf
      originated on PC's, and therefore they should dictate the
      standard.
    initialization of something predicting spaces changed 1990/04/22 by
      Paul Edwards so that when the compressed file is taken somewhere
      else, it will decode properly, without changing ascii spaces to
      ebcdic spaces.  This was done by changing the ' ' (space literal)
      to 0x20 (which is the far most likely character to occur, if you
      don't know what environment it will be running on.
**********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

FILE  *infile, *outfile;
static unsigned long int  textsize = 0, codesize = 0, printcount = 0;

char wterr[] = "Can't write.";

static void Error(char *message)
{
    printf("\n%s\n", message);
    exit(EXIT_FAILURE);
}

/********** LZSS compression **********/

#define N       4096    /* buffer size */
#define F       60  /* lookahead buffer size */
#define THRESHOLD   2
#define NIL     N   /* leaf of tree */

unsigned char
        text_buf[N + F - 1];
static int     match_position, match_length,
        lson[N + 1], rson[N + 257], dad[N + 1];

static void InitTree(void)  /* initialize trees */
{
    int  i;

    for (i = N + 1; i <= N + 256; i++)
        rson[i] = NIL;          /* root */
    for (i = 0; i < N; i++)
        dad[i] = NIL;           /* node */
}

static void InsertNode(int r)  /* insert to tree */
{
    int  i, p, cmp;
    unsigned char  *key;
    unsigned c;

    cmp = 1;
    key = &text_buf[r];
    p = N + 1 + key[0];
    rson[r] = lson[r] = NIL;
    match_length = 0;
    for ( ; ; ) {
        if (cmp >= 0) {
            if (rson[p] != NIL)
                p = rson[p];
            else {
                rson[p] = r;
                dad[r] = p;
                return;
            }
        } else {
            if (lson[p] != NIL)
                p = lson[p];
            else {
                lson[p] = r;
                dad[r] = p;
                return;
            }
        }
        for (i = 1; i < F; i++)
            if ((cmp = key[i] - text_buf[p + i]) != 0)
                break;
        if (i > THRESHOLD) {
            if (i > match_length) {
```

```
                match_position = ((r - p) & (N - 1)) - 1;
                if ((match_length = i) >= F)
                    break;
            }
            if (i == match_length) {
                if ((c = ((r - p) & (N-1)) - 1) < (unsigned)match_position) {
                    match_position = c;
                }
            }
        }
    }
    dad[r] = dad[p];
    lson[r] = lson[p];
    rson[r] = rson[p];
    dad[lson[p]] = r;
    dad[rson[p]] = r;
    if (rson[dad[p]] == p)
        rson[dad[p]] = r;
    else
        lson[dad[p]] = r;
    dad[p] = NIL; /* remove p */
}

static void DeleteNode(int p)  /* remove from tree */
{
    int  q;

    if (dad[p] == NIL)
        return;         /* not registered */
    if (rson[p] == NIL)
        q = lson[p];
    else
    if (lson[p] == NIL)
        q = rson[p];
    else {
        q = lson[p];
        if (rson[q] != NIL) {
            do {
                q = rson[q];
            } while (rson[q] != NIL);
            rson[dad[q]] = lson[q];
            dad[lson[q]] = dad[q];
            lson[q] = lson[p];
            dad[lson[p]] = q;
        }
        rson[q] = rson[p];
        dad[rson[p]] = q;
    }
    dad[q] = dad[p];
    if (rson[dad[p]] == p)
        rson[dad[p]] = q;
    else
        lson[dad[p]] = q;
    dad[p] = NIL;
}

/* Huffman coding */

#define N_CHAR      (256 - THRESHOLD + F)
                /* kinds of characters (character code = 0..N_CHAR-1) */
#define T       (N_CHAR * 2 - 1)    /* size of table */
#define R       (T - 1)         /* position of root */
#define MAX_FREQ    0x8000      /* updates tree when the */
typedef unsigned char uchar;


/* table for encoding and decoding the upper 6 bits of position */

/* for encoding */
uchar p_len[64] = {
    0x03, 0x04, 0x04, 0x04, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08
};

uchar p_code[64] = {
    0x00, 0x20, 0x30, 0x40, 0x50, 0x58, 0x60, 0x68,
    0x70, 0x78, 0x80, 0x88, 0x90, 0x94, 0x98, 0x9C,
    0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8, 0xBC,
    0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC, 0xCE,
    0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE,
    0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xEC, 0xEE,
    0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
    0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF
};

/* for decoding */
uchar d_code[256] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
```

```
    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
    0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09,
    0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A,
    0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B,
    0x0C, 0x0C, 0x0C, 0x0C, 0x0D, 0x0D, 0x0D, 0x0D,
    0x0E, 0x0E, 0x0E, 0x0E, 0x0F, 0x0F, 0x0F, 0x0F,
    0x10, 0x10, 0x10, 0x10, 0x11, 0x11, 0x11, 0x11,
    0x12, 0x12, 0x12, 0x12, 0x13, 0x13, 0x13, 0x13,
    0x14, 0x14, 0x14, 0x14, 0x15, 0x15, 0x15, 0x15,
    0x16, 0x16, 0x16, 0x16, 0x17, 0x17, 0x17, 0x17,
    0x18, 0x18, 0x19, 0x19, 0x1A, 0x1A, 0x1B, 0x1B,
    0x1C, 0x1C, 0x1D, 0x1D, 0x1E, 0x1E, 0x1F, 0x1F,
    0x20, 0x20, 0x21, 0x21, 0x22, 0x22, 0x23, 0x23,
    0x24, 0x24, 0x25, 0x25, 0x26, 0x26, 0x27, 0x27,
    0x28, 0x28, 0x29, 0x29, 0x2A, 0x2A, 0x2B, 0x2B,
    0x2C, 0x2C, 0x2D, 0x2D, 0x2E, 0x2E, 0x2F, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
};

uchar d_len[256] = {
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
};

unsigned freq[T + 1]; /* frequency table */

int prnt[T + N_CHAR]; /* pointers to parent nodes, except for the */
            /* elements [T..T + N_CHAR - 1] which are used to get */
            /* the positions of leaves corresponding to the codes. */

int son[T];   /* pointers to child nodes (son[], son[] + 1) */

unsigned getbuf = 0;
uchar getlen = 0;

static int GetBit(void)    /* get one bit */
{
    unsigned i;

    while (getlen <= 8) {
        if ((int)(i = getc(infile)) < 0) i = 0;
        getbuf |= i << (8 - getlen);
        getlen += 8;
    }
    i = getbuf;
    getbuf <<= 1;
    getlen--;
    return (int)((i & 0x8000) >> 15);
}

static int GetByte(void)   /* get one byte */
{
    unsigned i;

    while (getlen <= 8) {
        if ((int)(i = getc(infile)) < 0) i = 0;
        getbuf |= i << (8 - getlen);
        getlen += 8;
    }
    i = getbuf;
    getbuf <<= 8;
    getlen -= 8;
```

```
    return (int)((i & 0xff00) >> 8);
}

unsigned putbuf = 0;
uchar putlen = 0;

static void Putcode(int l, unsigned c)     /* output c bits of code */
{
    putbuf |= c >> putlen;
    if ((putlen += l) >= 8) {
        if (putc(putbuf >> 8, outfile) == EOF) {
            Error(wterr);
        }
        if ((putlen -= 8) >= 8) {
            if (putc(putbuf, outfile) == EOF) {
                Error(wterr);
            }
            codesize += 2;
            putlen -= 8;
            putbuf = c << (l - putlen);
        } else {
            putbuf <<= 8;
            codesize++;
        }
    }
}


/* initialization of tree */

static void StartHuff(void)
{
    int i, j;

    for (i = 0; i < N_CHAR; i++) {
        freq[i] = 1;
        son[i] = i + T;
        prnt[i + T] = i;
    }
    i = 0; j = N_CHAR;
    while (j <= R) {
        freq[j] = freq[i] + freq[i + 1];
        son[j] = i;
        prnt[i] = prnt[i + 1] = j;
        i += 2; j++;
    }
    freq[T] = 0xffff;
    prnt[R] = 0;
}


/* reconstruction of tree */

static void reconst(void)
{
    int i, j, k;
    unsigned f, l;

    /* collect leaf nodes in the first half of the table */
    /* and replace the freq by (freq + 1) / 2. */
    j = 0;
    for (i = 0; i < T; i++) {
        if (son[i] >= T) {
            freq[j] = (freq[i] + 1) / 2;
            son[j] = son[i];
            j++;
        }
    }
    /* begin constructing tree by connecting sons */
    for (i = 0, j = N_CHAR; j < T; i += 2, j++) {
        k = i + 1;
        f = freq[j] = freq[i] + freq[k];
        for (k = j - 1; f < freq[k]; k--);
        k++;
        l = (j - k) * 2;
        memmove(&freq[k + 1], &freq[k], l);
        freq[k] = f;
        memmove(&son[k + 1], &son[k], l);
        son[k] = i;
    }
    /* connect prnt */
    for (i = 0; i < T; i++) {
        if ((k = son[i]) >= T) {
            prnt[k] = i;
        } else {
            prnt[k] = prnt[k + 1] = i;
        }
    }
}


/* increment frequency of given code by one, and update tree */

static void update(int c)
{
    int i, j, k, l;

    if (freq[R] == MAX_FREQ) {
        reconst();
    }
```

```
401 |     c = prnt[c + T];
402 |     do {
403 |         k = ++freq[c];
404 |
405 |         /* if the order is disturbed, exchange nodes */
406 |         if ((unsigned)k > freq[l = c + 1]) {
407 |             while ((unsigned)k > freq[++l]);
408 |             l--;
409 |             freq[c] = freq[l];
410 |             freq[l] = k;
411 |
412 |             i = son[c];
413 |             prnt[i] = l;
414 |             if (i < T) prnt[i + 1] = l;
415 |
416 |             j = son[l];
417 |             son[l] = i;
418 |
419 |             prnt[j] = c;
420 |             if (j < T) prnt[j + 1] = c;
421 |             son[c] = j;
422 |
423 |             c = l;
424 |         }
425 |     } while ((c = prnt[c]) != 0); /* repeat up to root */
426 | }
427 |
428 | unsigned code, len;
429 |
430 | static void EncodeChar(unsigned c)
431 | {
432 |     unsigned i;
433 |     int j, k;
434 |
435 |     i = 0;
436 |     j = 0;
437 |     k = prnt[c + T];
438 |
439 |     /* travel from leaf to root */
440 |     do {
441 |         i >>= 1;
442 |
443 |         /* if node's address is odd-numbered, choose bigger brother node */
444 |         if (k & 1) i += 0x8000;
445 |
446 |         j++;
447 |     } while ((k = prnt[k]) != R);
448 |     Putcode(j, i);
449 |     code = i;
450 |     len = j;
451 |     update(c);
452 | }
453 |
454 | static void EncodePosition(unsigned c)
455 | {
456 |     unsigned i;
457 |
458 |     /* output upper 6 bits by table lookup */
459 |     i = c >> 6;
460 |     Putcode(p_len[i], (unsigned)p_code[i] << 8);
461 |
462 |     /* output lower 6 bits verbatim */
463 |     Putcode(6, (c & 0x3f) << 10);
464 | }
465 |
466 | static void EncodeEnd(void)
467 | {
468 |     if (putlen) {
469 |         if (putc(putbuf >> 8, outfile) == EOF) {
470 |             Error(wterr);
471 |         }
472 |         codesize++;
473 |     }
474 | }
475 |
476 | static int DecodeChar(void)
477 | {
478 |     unsigned c;
479 |
480 |     c = son[R];
481 |
482 |     /* travel from root to leaf, */
483 |     /* choosing the smaller child node (son[]) if the read bit is 0, */
484 |     /* the bigger (son[]+1} if 1 */
485 |     while (c < T) {
486 |         c += GetBit();
487 |         c = son[c];
488 |     }
489 |     c -= T;
490 |     update(c);
491 |     return (int)c;
492 | }
493 |
494 | static int DecodePosition(void)
495 | {
496 |     unsigned i, j, c;
497 |
498 |     /* recover upper 6 bits from table */
499 |     i = GetByte();
500 |     c = (unsigned)d_code[i] << 6;
```

```
    j = d_len[i];

    /* read lower 6 bits verbatim */
    j -= 2;
    while (j--) {
        i = (i << 1) + GetBit();
    }
    return (int)(c | (i & 0x3f));
}

/* compression */

static void Encode(void)  /* compression */
{
    int  i, c, len, r, s, last_match_length;

    fseek(infile, 0L, 2);
    textsize = ftell(infile);
    fputc((int)((textsize & 0xff)),outfile);
    fputc((int)((textsize & 0xff00) >> 8),outfile);
    fputc((int)((textsize & 0xff0000L) >> 16),outfile);
    fputc((int)((textsize & 0xff000000L) >> 24),outfile);
    if (ferror(outfile))
        Error(wterr);   /* output size of text */
    if (textsize == 0)
        return;
    rewind(infile);
    textsize = 0;           /* rewind and re-read */
    StartHuff();
    InitTree();
    s = 0;
    r = N - F;
    for (i = s; i < r; i++)
        text_buf[i] = 0x20;
    for (len = 0; len < F && (c = getc(infile)) != EOF; len++)
        text_buf[r + len] = (unsigned char)c;
    textsize = len;
    for (i = 1; i <= F; i++)
        InsertNode(r - i);
    InsertNode(r);
    do {
        if (match_length > len)
            match_length = len;
        if (match_length <= THRESHOLD) {
            match_length = 1;
            EncodeChar(text_buf[r]);
        } else {
            EncodeChar(255 - THRESHOLD + match_length);
            EncodePosition(match_position);
        }
        last_match_length = match_length;
        for (i = 0; i < last_match_length &&
                (c = getc(infile)) != EOF; i++) {
            DeleteNode(s);
            text_buf[s] = (unsigned char)c;
            if (s < F - 1)
                text_buf[s + N] = (unsigned char)c;
            s = (s + 1) & (N - 1);
            r = (r + 1) & (N - 1);
            InsertNode(r);
        }
        if ((textsize += i) > printcount) {
            printf("%12ld\r", textsize);
            printcount += 1024;
        }
        while (i++ < last_match_length) {
            DeleteNode(s);
            s = (s + 1) & (N - 1);
            r = (r + 1) & (N - 1);
            if (--len) InsertNode(r);
        }
    } while (len > 0);
    EncodeEnd();
    printf("In : %ld bytes\n", textsize);
    printf("Out: %ld bytes\n", codesize);
    printf("Out/In: %.3f\n", 1.0 * codesize / textsize);
}

static void Decode(void)  /* recover */
{
    int  i, j, k, r, c;
    unsigned long int  count;

    textsize = (fgetc(infile));
    textsize |= ((unsigned long)fgetc(infile) << 8);
    textsize |= ((unsigned long)fgetc(infile) << 16);
    textsize |= ((unsigned long)fgetc(infile) << 24);
    if (ferror(infile))
        Error("Can't read");  /* read size of text */
    if (textsize == 0)
        return;
    StartHuff();
    for (i = 0; i < N - F; i++)
        text_buf[i] = 0x20;
    r = N - F;
    for (count = 0; count < textsize; ) {
        c = DecodeChar();
        if (c < 256) {
            if (putc(c, outfile) == EOF) {
                Error(wterr);
```

```
            }
            text_buf[r++] = (unsigned char)c;
            r &= (N - 1);
            count++;
        } else {
            i = (r - DecodePosition() - 1) & (N - 1);
            j = c - 255 + THRESHOLD;
            for (k = 0; k < j; k++) {
                c = text_buf[(i + k) & (N - 1)];
                if (putc(c, outfile) == EOF) {
                    Error(wterr);
                }
                text_buf[r++] = (unsigned char)c;
                r &= (N - 1);
                count++;
            }
        }
        if (count > printcount) {
            printf("%12ld\r", count);
            printcount += 1024;
        }
    }
    printf("%12ld\n", count);
}

int main(int argc, char *argv[])
{
    char  *s;

    if (argc != 4) {
        printf("'lzhuf e file1 file2' encodes file1 into file2.\n"
               "'lzhuf d file2 file1' decodes file2 into file1.\n");
        return EXIT_FAILURE;
    }
    if ((s = argv[1], s[1] || strpbrk(s, "DEde") == NULL)
     || (s = argv[2], (infile = fopen(s, "rb")) == NULL)
     || (s = argv[3], (outfile = fopen(s, "wb")) == NULL)) {
        printf("??? %s\n", s);
        return EXIT_FAILURE;
    }
    if (toupper(*argv[1]) == 'E')
        Encode();
    else
        Decode();
    fclose(infile);
    fclose(outfile);
    return EXIT_SUCCESS;
}
```

TEXT STATISTICS

19006 characters
648 lines

LEXICAL STATISTICS

53 comments [std-C]
0 comments [C++]
12 preprocessor instructions
1 constants [character]
15 constants [string]
794 constants [numeric]

SYMBOL TABLE

```
    Decode          579     644
    DecodeChar              476     597
    DecodePosition          494     606
    DeleteNode              124     554     567
    EOF             311     315     469     535     553     599     610
    EXIT_FAILURE             42     633     639
    EXIT_SUCCESS            647
    Encode          513     642
    EncodeChar              430     546     548
    EncodeEnd       466     573
    EncodePosition          454     549
    Error            39     312     316     470     524     589     600     611
    F        48      53      96     102     159     532     535     538     556     593     595
    FILE             34
    GetBit          274     486     506
    GetByte         289     499
    InitTree         57     530
    InsertNode               67     539     540     560     570
    MAX_FREQ        163     398
    N        47      50      53      55+     61+     63      75     101     106     532     557     558
           559     568     569     593     595     603     606     609     614
    NIL              50      62      64      76      80      88     121     128     130     133     137
           140     154
    NULL            635     636     637
    N_CHAR          159     161     265     335     340     370
    Putcode         307     448     460     463
    R       162     341     348     398     447     480
    StartHuff       331     529     592
    T       161     162     263     265     269     337     338     347     362     363     370     382
           383     401     414     420     437     485     489
    THRESHOLD        49      99     159     544     548     607
    argc            626     630
    argv            626     635     636     637     641
    c        71     106     107     307     309     320     394     401+    403     406     409     412
           419     420     421     423     425+    430     437     451     454     459     463     478
           480     485     486     487+    489     490     491     496     500     508     515     535
           536     553     555     557     581     597     598     599     602     607     609     610
           613
    cmp              69      73      79      97
    code            428     449
    codesize         35     318     323     472     575     576
    count           582     596+    604     615     618     619     623
    ctype            32
    d_code          193     500
    d_len           228     501
    dad              55      64      84      92     112+    115     116     117     118     120     121
           128     141     142+    144     147     149+    150     151     153     154
    exit             42
    f       357     372     373     377
    fclose          645     646
    ferror          523     588
    fgetc           584     585     586     587
    fopen           636     637
    fputc           519     520     521     522
    freq            263     336     342+    347     364+    372+    373     376+    377     398     403
           406     407     409+    410
    fseek           517
    ftell           518
    getbuf          271     280     283     284     295     298     299
    getc            279     294     535     553
    getlen          272     278     280     281     285     293     295     296     300
    h        29      30      31      32
    i        59      61+     62      63+     64      69      96+     97+     99     100     102     105
           276     279+    280     283     286     291     294+    295     298     301     333     335+
           336     337+    338+    340     342+    343     344+    345     356     362+    363     364
           365     370+    371     372     379     382+    383     384     386     396     412     413
           414+    417     432     435     441     444     448     449     456     459     460+    496
           499     500     501     506+    508     515     533+    534     538+    539     552+    553
           562     566     581     593+    594     606     609
    infile           34     279     294     517     518     527     535     553     584     585     586
           587     588     636     645
    j       333     340     341     342     343     344     345     356     361     364     365     366
           370+    372     373     375     396     416     419     420+    421     433     436     446
           448     450     496     501     504     505     581     607     608
    k       356     371     372     373+    374     375     376+    377     378+    379     383     384
           386+    396     403     406     407     410     433     437     444     447+    581     608+
           609
    key              70      74      75      97
    l       307     310     320     357     375     376     378     396     406     407     408     409
           410     413     414     416     417     423
    last_match_length       515     551     552     566
    len             428     450     515     535+    536     537     542     543     570     572
    lson             55      76      88      89      91     113+    115     120     131     133     136
           141     142     143+    144     153
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | main(){char *c="main(){char *c=%c%s%c;printf(c,34,c,34);}";printf(c,34,c,34);}
```

TEXT STATISTICS

```
   124 characters
     3 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    c        3+
    main             3
    printf           3
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 EPSHeader

   File: match.c
   Author: J. Kercheval
   Created: Sat, 01/05/1991  22:21:49
*/
/*
 EPSRevision History

   J. Kercheval  Wed, 02/20/1991  22:29:01  Released to Public Domain
   J. Kercheval  Fri, 02/22/1991  15:29:01  fix '\' bugs (two :( of them)
   J. Kercheval  Sun, 03/10/1991  19:31:29  add error return to matche()
   J. Kercheval  Sun, 03/10/1991  20:11:11  add is_valid_pattern code
   J. Kercheval  Sun, 03/10/1991  20:37:11  beef up main()
   J. Kercheval  Tue, 03/12/1991  22:25:10  Released as V1.1 to Public Domain
*/

/*
   Wildcard Pattern Matching
*/


#include "match.h"

int matche_after_star (register char *pattern, register char *text);
int fast_match_after_star (register char *pattern, register char *text);

/*---------------------------------------------------------------------------
*
* Return TRUE if PATTERN has any special wildcard characters
*
---------------------------------------------------------------------------*/

BOOLEAN is_pattern (char *p)
{
      while (*p)
      {
            switch (*p++)
            {
            case '?':
            case '*':
            case '[':
            case '\\':
                  return TRUE;
            }
      }
      return FALSE;
}


/*---------------------------------------------------------------------------
*
* Return TRUE if PATTERN has is a well formed regular expression according
* to the above syntax
*
* error_type is a return code based on the type of pattern error.  Zero is
* returned in error_type if the pattern is a valid one.  error_type return
* values are as follows:
*
*   PATTERN_VALID - pattern is well formed
*   PATTERN_ESC   - pattern has invalid escape ('\' at end of pattern)
*   PATTERN_RANGE - [..] construct has a no end range in a '-' pair (ie [a-])
*   PATTERN_CLOSE - [..] construct has no end bracket (ie [abc-g )
*   PATTERN_EMPTY - [..] construct is empty (ie [])
*
---------------------------------------------------------------------------*/

BOOLEAN is_valid_pattern (char *p, int *error_type)
{
      /* init error_type */
      *error_type = PATTERN_VALID;

      /* loop through pattern to EOS */
      while (*p)
      {
            /* determine pattern type */
            switch (*p)
            {
            /* check literal escape, it cannot be at end of pattern */
            case '\\':
                  if (!*++p)
                  {
                        *error_type = PATTERN_ESC;
                        return FALSE;
                  }
                  p++;
                  break;

                  /* the [..] construct must be well formed */
            case '[':
                  p++;

                  /* if the next character is ']' then bad pattern */
                  if (*p == ']')
                  {
                        *error_type = PATTERN_EMPTY;
                        return FALSE;
```

```
                }

                /* if end of pattern here then bad pattern */
                if (!*p)
                {
                      *error_type = PATTERN_CLOSE;
                      return FALSE;
                }

                /* loop to end of [..] construct */
                while (*p != ']')
                {
                      /* check for literal escape */
                      if (*p == '\\')
                      {
                            p++;

                            /* if end of pattern here then bad pattern */
                            if (!*p++)
                            {
                                  *error_type = PATTERN_ESC;
                                  return FALSE;
                            }
                      }
                      else  p++;

                      /* if end of pattern here then bad pattern */
                      if (!*p)
                      {
                            *error_type = PATTERN_CLOSE;
                            return FALSE;
                      }

                      /* if this a range */
                      if (*p == '-')
                      {
                            /* we must have an end of range */
                            if (!*++p || *p == ']')
                            {
                                  *error_type = PATTERN_RANGE;
                                  return FALSE;
                            }
                            else
                            {

                                  /* check for literal escape */
                                  if (*p == '\\')
                                        p++;

                                  /* if end of pattern here
                                     then bad pattern           */
                                  if (!*p++)
                                  {
                                        *error_type = PATTERN_ESC;
                                        return FALSE;
                                  }
                            }
                      }
                }
                break;

                /* all other characters are valid pattern elements */
          case '*':
          case '?':
          default:
                p++;                              /* "normal" character */
                break;
          }
     }
     return TRUE;
}


/*---------------------------------------------------------------------------
 *
 *  Match the pattern PATTERN against the string TEXT;
 *
 *  returns MATCH_VALID if pattern matches, or an errorcode as follows
 *  otherwise:
 *
 *            MATCH_PATTERN  - bad pattern
 *            MATCH_LITERAL  - match failure on literal mismatch
 *            MATCH_RANGE    - match failure on [..] construct
 *            MATCH_ABORT    - premature end of text string
 *            MATCH_END      - premature end of pattern string
 *            MATCH_VALID    - valid match
 *
 *
 *  A match means the entire string TEXT is used up in matching.
 *
 *  In the pattern string:
 *       `*' matches any sequence of characters (zero or more)
 *       `?' matches any character
 *       [SET] matches any character in the specified set,
 *       [!SET] or [^SET] matches any character not in the specified set.
 *
 *  A set is composed of characters or ranges; a range looks like
 *  character hyphen character (as in 0-9 or A-Z).  [0-9a-zA-Z_] is the
 *  minimal set of characters allowed in the [..] pattern construct.
 *  Other characters are allowed (ie. 8 bit characters) if your system
```

```
201 | *  will support them.
202 | *
203 | *  To suppress the special syntactic significance of any of `[]*?!^-\',
204 | *  and match the character exactly, precede it with a `\'.
205 | *
206 | ------------------------------------------------------------------------*/
207 |
208 | int matche (register char *p, register char *t)
209 | {
210 |       register char range_start, range_end;  /* start and end in range */
211 |
212 |       BOOLEAN invert;             /* is this [..] or [!..] */
213 |       BOOLEAN member_match;       /* have I matched the [..] construct? */
214 |       BOOLEAN loop;               /* should I terminate? */
215 |
216 |       for ( ; *p; p++, t++)
217 |       {
218 |             /* if this is the end of the text
219 |                then this is the end of the match */
220 |
221 |             if (!*t)
222 |             {
223 |                   return ( *p == '*' && *++p == '\0' ) ?
224 |                         MATCH_VALID : MATCH_ABORT;
225 |             }
226 |
227 |             /* determine and react to pattern type */
228 |
229 |             switch (*p)
230 |             {
231 |             case '?':                     /* single any character match */
232 |                   break;
233 |
234 |             case '*':                     /* multiple any character match */
235 |                   return matche_after_star (p, t);
236 |
237 |             /* [..] construct, single member/exclusion character match */
238 |             case '[':
239 |             {
240 |                   /* move to beginning of range */
241 |
242 |                   p++;
243 |
244 |                   /* check if this is a member match or exclusion match */
245 |
246 |                   invert = FALSE;
247 |                   if (*p == '!' || *p == '^')
248 |                   {
249 |                         invert = TRUE;
250 |                         p++;
251 |                   }
252 |
253 |                   /* if closing bracket here or at range start then we have a
254 |                         malformed pattern */
255 |
256 |                   if (*p == ']')
257 |                   {
258 |                         return MATCH_PATTERN;
259 |                   }
260 |
261 |                   member_match = FALSE;
262 |                   loop = TRUE;
263 |
264 |                   while (loop)
265 |                   {
266 |                         /* if end of construct then loop is done */
267 |
268 |                         if (*p == ']')
269 |                         {
270 |                               loop = FALSE;
271 |                               continue;
272 |                         }
273 |
274 |                         /* matching a '!', '^', '-', '\' or a ']' */
275 |
276 |                         if (*p == '\\')
277 |                         {
278 |                               range_start = range_end = *++p;
279 |                         }
280 |                         else  range_start = range_end = *p;
281 |
282 |                         /* if end of pattern then bad pattern (Missing ']') */
283 |
284 |                         if (!*p)
285 |                               return MATCH_PATTERN;
286 |
287 |                         /* check for range bar */
288 |                         if (*++p == '-')
289 |                         {
290 |                               /* get the range end */
291 |
292 |                               range_end = *++p;
293 |
294 |                               /* if end of pattern or construct
295 |                                  then bad pattern */
296 |
297 |                               if (range_end == '\0' || range_end == ']')
298 |                                     return MATCH_PATTERN;
299 |
300 |                               /* special character range end */
```

```
301 |                                 if (range_end == '\\')
302 |                                 {
303 |                                       range_end = *++p;
304 |
305 |                                       /* if end of text then
306 |                                          we have a bad pattern */
307 |                                       if (!range_end)
308 |                                             return MATCH_PATTERN;
309 |                                 }
310 |
311 |                                 /* move just beyond this range */
312 |                                 p++;
313 |                           }
314 |
315 |                           /* if the text character is in range then match found.
316 |                              make sure the range letters have the proper
317 |                              relationship to one another before comparison */
318 |
319 |                           if (range_start < range_end)
320 |                           {
321 |                                 if (*t >= range_start && *t <= range_end)
322 |                                 {
323 |                                       member_match = TRUE;
324 |                                       loop = FALSE;
325 |                                 }
326 |                           }
327 |                           else
328 |                           {
329 |                                 if (*t >= range_end && *t <= range_start)
330 |                                 {
331 |                                       member_match = TRUE;
332 |                                       loop = FALSE;
333 |                                 }
334 |                           }
335 |                     }
336 |
337 |                     /* if there was a match in an exclusion set then no match */
338 |                     /* if there was no match in a member set then no match */
339 |
340 |                     if ((invert && member_match) || !(invert || member_match))
341 |                           return MATCH_RANGE;
342 |
343 |                     /* if this is not an exclusion then skip the rest of
344 |                        the [...] construct that already matched. */
345 |
346 |                     if (member_match)
347 |                     {
348 |                           while (*p != ']')
349 |                           {
350 |                                 /* bad pattern (Missing ']') */
351 |                                 if (!*p)
352 |                                       return MATCH_PATTERN;
353 |
354 |                                 /* skip exact match */
355 |                                 if (*p == '\\')
356 |                                 {
357 |                                       p++;
358 |
359 |                                       /* if end of text then
360 |                                          we have a bad pattern */
361 |
362 |                                       if (!*p)
363 |                                             return MATCH_PATTERN;
364 |                                 }
365 |
366 |                                 /* move to next pattern char */
367 |
368 |                                 p++;
369 |                           }
370 |                     }
371 |                     break;
372 |               }
373 |               case '\\':  /* next character is quoted and must match exactly */
374 |
375 |                     /* move pattern pointer to quoted char and fall through */
376 |
377 |                     p++;
378 |
379 |                     /* if end of text then we have a bad pattern */
380 |
381 |                     if (!*p)
382 |                           return MATCH_PATTERN;
383 |
384 |                     /* must match this character exactly */
385 |
386 |               default:
387 |                     if (*p != *t)
388 |                           return MATCH_LITERAL;
389 |               }
390 |         }
391 |         /* if end of text not reached then the pattern fails */
392 |
393 |         if (*t)
394 |               return MATCH_END;
395 |         else  return MATCH_VALID;
396 | }
397 |
398 |
399 | /*----------------------------------------------------------------------
400 | *
```

```
401 | * recursively call matche() with final segment of PATTERN and of TEXT.
402 | *
403 | ------------------------------------------------------------------------*/
404 |
405 | int matche_after_star (register char *p, register char *t)
406 | {
407 |       register int match = 0;
408 |       register nextp;
409 |
410 |       /* pass over existing ? and * in pattern */
411 |
412 |       while ( *p == '?' || *p == '*' )
413 |       {
414 |             /* take one char for each ? and + */
415 |
416 |             if (*p == '?')
417 |             {
418 |                   /* if end of text then no match */
419 |                   if (!*t++)
420 |                         return MATCH_ABORT;
421 |             }
422 |
423 |             /* move to next char in pattern */
424 |
425 |             p++;
426 |       }
427 |
428 |       /* if end of pattern we have matched regardless of text left */
429 |
430 |       if (!*p)
431 |             return MATCH_VALID;
432 |
433 |       /* get the next character to match which must be a literal or '[' */
434 |
435 |       nextp = *p;
436 |       if (nextp == '\\')
437 |       {
438 |             nextp = p[1];
439 |
440 |             /* if end of text then we have a bad pattern */
441 |
442 |             if (!nextp)
443 |                   return MATCH_PATTERN;
444 |       }
445 |
446 |       /* Continue until we run out of text or definite result seen */
447 |
448 |       do
449 |       {
450 |             /* a precondition for matching is that the next character
451 |                in the pattern match the next character in the text or that
452 |                the next pattern char is the beginning of a range.  Increment
453 |                text pointer as we go here */
454 |
455 |             if (nextp == *t || nextp == '[')
456 |                   match = matche(p, t);
457 |
458 |             /* if the end of text is reached then no match */
459 |
460 |             if (!*t++)
461 |                   match = MATCH_ABORT;
462 |
463 |       } while ( match != MATCH_VALID &&
464 |                  match != MATCH_ABORT &&
465 |                  match != MATCH_PATTERN);
466 |
467 |       /* return result */
468 |
469 |       return match;
470 | }
471 |
472 |
473 | /*------------------------------------------------------------------------
474 | *
475 | * match() is a shell to matche() to return only BOOLEAN values.
476 | *
477 | ------------------------------------------------------------------------*/
478 |
479 | BOOLEAN match( char *p, char *t )
480 | {
481 |       int error_type;
482 |
483 |       error_type = matche(p,t);
484 |       return (error_type == MATCH_VALID ) ? TRUE : FALSE;
485 | }
486 |
487 |
488 | #ifdef TEST
489 |
490 | /*
491 | ** This test main expects as first arg the pattern and as second arg
492 | ** the match string.  Output is yea or nay on match.  If nay on
493 | ** match then the error code is parsed and written.
494 | */
495 |
496 | #include <stdio.h>
497 |
498 | int main(int argc, char *argv[])
499 | {
500 |       int error;
```

```
501 |       int is_valid_error;
502 |
503 |       if (argc != 3)
504 |             printf("Usage:  MATCH Pattern Text\n");
505 |       else
506 |       {
507 |             printf("Pattern: %s\n", argv[1]);
508 |             printf("Text   : %s\n", argv[2]);
509 |
510 |             if (!is_pattern(argv[1]))
511 |                   printf("    First Argument Is Not A Pattern\n");
512 |             else
513 |             {
514 |                   error = matche(argv[1],argv[2]);
515 |                   is_valid_pattern(argv[1],&is_valid_error);
516 |
517 |                   switch (error)
518 |                   {
519 |                   case MATCH_VALID:
520 |                         printf("    Match Successful");
521 |                         if (is_valid_error != PATTERN_VALID)
522 |                               printf(" -- is_valid_pattern() "
523 |                                     "is complaining\n");
524 |                         else  printf("\n");
525 |                         break;
526 |
527 |                   case MATCH_LITERAL:
528 |                         printf("    Match Failed on Literal\n");
529 |                         break;
530 |
531 |                   case MATCH_RANGE:
532 |                         printf("    Match Failed on [..]\n");
533 |                         break;
534 |
535 |                   case MATCH_ABORT:
536 |                         printf("    Match Failed on Early "
537 |                               "Text Termination\n");
538 |                         break;
539 |
540 |                   case MATCH_END:
541 |                         printf("    Match Failed on Early "
542 |                               "Pattern Termination\n");
543 |                         break;
544 |
545 |                   case MATCH_PATTERN:
546 |                         switch (is_valid_error)
547 |                         {
548 |                         case PATTERN_VALID:
549 |                               printf("    Internal Disagreement "
550 |                                     "On Pattern\n");
551 |                               break;
552 |
553 |                         case PATTERN_ESC:
554 |                               printf("    Literal Escape at "
555 |                                     "End of Pattern\n");
556 |                               break;
557 |
558 |
559 |                         case PATTERN_RANGE:
560 |                               printf("    No End of Range in "
561 |                                     "[..] Construct\n");
562 |                               break;
563 |
564 |                         case PATTERN_CLOSE:
565 |                               printf("    [..] Construct is Open\n");
566 |                               break;
567 |
568 |                         case PATTERN_EMPTY:
569 |                               printf("    [..] Construct is Empty\n");
570 |                               break;
571 |
572 |                         default:
573 |                               printf("    Internal Error in "
574 |                                     "is_valid_pattern()\n");
575 |                         }
576 |                         break;
577 |
578 |                   default:
579 |                         printf("    Internal Error in matche()\n");
580 |                         break;
581 |                   }
582 |             }
583 |       }
584 |       return(0);
585 | }
586 |
587 | #endif /* TEST */
```

TEXT STATISTICS

```
 19394 characters
   587 lines
```

LEXICAL STATISTICS

```
    73 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
    36 constants [character]
    26 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    BOOLEAN          37      71     212     213     214     479
    FALSE            50      87     100     107     122     131     141     155     246     261     270
           324     332     484
    MATCH_ABORT              224     420     461     464     535
    MATCH_END       394     540
    MATCH_LITERAL            388     527
    MATCH_PATTERN            258     285     298     308     352     363     382     443     465     545
    MATCH_RANGE              341     531
    MATCH_VALID              224     395     431     463     484     519
    PATTERN_CLOSE            106     130     564
    PATTERN_EMPTY             99     568
    PATTERN_ESC               86     121     154     553
    PATTERN_RANGE            140     559
    PATTERN_VALID             74     521     548
    TEST            488
    TRUE              47     170     249     262     323     331     484
    argc            498     503
    argv            498     507     508     510     514+    515
    error           500     514     517
    error_type                71      74      86      99     106     121     130     140     154     481
           483     484
    fast_match_after_star             29
    h       496
    invert          212     246     249     340+
    is_pattern                37     510
    is_valid_error           501     515     521     546
    is_valid_pattern          71     515
    loop            214     262     264     270     324     332
    main            498
    match           407     456     461     463     464     465     469     479
    matche          208     456     483     514
    matche_after_star         28     235     405
    member_match             213     261     323     331     340+    346
    nextp           408     435     436     438     442     455+
    p        37      39      41      71      77      80      84      89      94      97     104     111
           114     116     119     125     128     135     138+    147     148     152     166     208
           216+    223+    229     235     242     247+    250     256     268     276     278     280
           284     288     292     303     312     348     351     355     357     362     368     377
           381     387     405     412+    416     425     430     435     438     456     479     483
    pattern           28      29
    printf          504     507     508     511     520     522     524     528     532     536     541
           549     554     560     565     569     573     579
    range_end       210     278     280     292     297+    301     303     307     319     321     329
    range_start              210     278     280     319     321     329
    stdio           496
    t       208     216     221     235     321+    329+    387     393     405     419     455     456
           460     479     483
    text             28      29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 EPSHeader

   File: match.h
   Author: J. Kercheval
   Created: Sat, 01/05/1991  22:27:18
*/
/*
 EPSRevision History

   J. Kercheval  Wed, 02/20/1991  22:28:37  Released to Public Domain
   J. Kercheval  Sun, 03/10/1991  18:02:56  add is_valid_pattern
   J. Kercheval  Sun, 03/10/1991  18:25:48  add error_type in is_valid_pattern
   J. Kercheval  Sun, 03/10/1991  18:47:47  error return from matche()
   J. Kercheval  Tue, 03/12/1991  22:24:49  Released as V1.1 to Public Domain
*/

#ifndef MATCH__H
#define MATCH__H

/*
   Wildcard Pattern Matching
*/

#ifndef BOOLEAN
# define BOOLEAN int
# define TRUE 1
# define FALSE 0
#endif

/* match defines */
#define MATCH_PATTERN  6    /* bad pattern */
#define MATCH_LITERAL  5    /* match failure on literal match */
#define MATCH_RANGE    4    /* match failure on [..] construct */
#define MATCH_ABORT    3    /* premature end of text string */
#define MATCH_END      2    /* premature end of pattern string */
#define MATCH_VALID    1    /* valid match */

/* pattern defines */
#define PATTERN_VALID  0    /* valid pattern */
#define PATTERN_ESC   -1    /* literal escape at end of pattern */
#define PATTERN_RANGE -2    /* malformed range in [..] construct */
#define PATTERN_CLOSE -3    /* no end bracket in [..] construct */
#define PATTERN_EMPTY -4    /* [..] construct is empty */

/*--------------------------------------------------------------------------
*
*  Match the pattern PATTERN against the string TEXT;
*
*       match() returns TRUE if pattern matches, FALSE otherwise.
*       matche() returns MATCH_VALID if pattern matches, or an errorcode
*           as follows otherwise:
*
*            MATCH_PATTERN  - bad pattern
*            MATCH_LITERAL  - match failure on literal mismatch
*            MATCH_RANGE    - match failure on [..] construct
*            MATCH_ABORT    - premature end of text string
*            MATCH_END      - premature end of pattern string
*            MATCH_VALID    - valid match
*
*
*  A match means the entire string TEXT is used up in matching.
*
*  In the pattern string:
*       `*' matches any sequence of characters (zero or more)
*       `?' matches any character
*       [SET] matches any character in the specified set,
*       [!SET] or [^SET] matches any character not in the specified set.
*
*  A set is composed of characters or ranges; a range looks like
*  character hyphen character (as in 0-9 or A-Z).  [0-9a-zA-Z_] is the
*  minimal set of characters allowed in the [..] pattern construct.
*  Other characters are allowed (ie. 8 bit characters) if your system
*  will support them.
*
*  To suppress the special syntactic significance of any of `[]*?!^-\',
*  and match the character exactly, precede it with a `\'.
*
--------------------------------------------------------------------------*/

BOOLEAN match (char *pattern, char *text);

int     matche(register char *pattern, register char *text);

/*--------------------------------------------------------------------------
*
* Return TRUE if PATTERN has any special wildcard characters
*
--------------------------------------------------------------------------*/

BOOLEAN is_pattern (char *pattern);

/*--------------------------------------------------------------------------
*
* Return TRUE if PATTERN has is a well formed regular expression according
* to the above syntax
*
* error_type is a return code based on the type of pattern error.  Zero is
```

```
 101 | * returned in error_type if the pattern is a valid one.  error_type return
 102 | * values are as follows:
 103 | *
 104 | *   PATTERN_VALID - pattern is well formed
 105 | *   PATTERN_ESC   - pattern has invalid escape ('\' at end of pattern)
 106 | *   PATTERN_RANGE - [..] construct has a no end range in a '-' pair (ie [a-])
 107 | *   PATTERN_CLOSE - [..] construct has no end bracket (ie [abc-g )
 108 | *   PATTERN_EMPTY - [..] construct is empty (ie [])
 109 | *
 110 | ----------------------------------------------------------------------------*/
 111 |
 112 | BOOLEAN is_valid_pattern (char *pattern, int *error_type);
 113 |
 114 | #endif /* MATCH__H */
```

TEXT STATISTICS

```
  4176 characters
   114 lines
```

LEXICAL STATISTICS

```
    21 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     0 constants [character]
     0 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    BOOLEAN         27      28      83      93     112
    FALSE           30
    MATCH_ABORT             37
    MATCH_END       38
    MATCH_LITERAL           35
    MATCH_PATTERN           34
    MATCH_RANGE             36
    MATCH_VALID             39
    MATCH__H        20      21
    PATTERN_CLOSE           45
    PATTERN_EMPTY           46
    PATTERN_ESC             43
    PATTERN_RANGE           44
    PATTERN_VALID           42
    TRUE            29
    define          28      29      30
    error_type             112
    is_pattern              93
    is_valid_pattern       112
    match           83
    matche          85
    pattern         83      85      93     112
    text            83      85
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MATHSTAT.C - Statistical analysis in C and C++
**
**  Public domain by Bob Stout
*/

#include <stddef.h>
#include <float.h>
#include "mathstat.h"

/*
**  Initialize a stat structure
*/

#if !(__cplusplus)

void stat_init(Stat_T *ptr)
{
      ptr->count   = 0;
      ptr->total   = 0.0;
      ptr->total2  = 0.0;
      ptr->recip   = 0.0;
      ptr->product = 1.0;
      ptr->min1 = ptr->min2 = ptr->oldmin = DBL_MAX;
      ptr->max1 = ptr->max2 = ptr->oldmax = DBL_MIN;
}

#else /* C++ */

inline Stat::Stat()
{
      count   = 0;
      total   = 0.0;
      total2  = 0.0;
      recip   = 0.0;
      product = 1.0;
      min1 = min2 = oldmin = DBL_MAX;
      max1 = max2 = oldmax = DBL_MIN;
}

#endif

/*
**  Return the number of data points
*/

#if !(__cplusplus)

size_t stat_count(Stat_T *ptr)
{
      return ptr->count;
}

#else /* C++ */

inline size_t Stat::Count()
{
      return count;
}

#endif

/*
**  Add a value for statistical analysis
*/

#if !(__cplusplus)

size_t stat_add(double datum, Stat_T *ptr)
{
      ptr->total   += datum;
      ptr->total2  += (datum * datum);
      ptr->recip   += 1.0 / datum;
      ptr->product *= datum;
      ++(ptr->count);
      if (datum < ptr->min1)
      {
            ptr->oldmin = ptr->min1;
            ptr->min1 = datum;
      }
      else if (datum < ptr->min2)
            ptr->min2 = datum;
      if (datum > ptr->max1)
      {
            ptr->oldmax = ptr->max1;
            ptr->max1 = datum;
      }
      else if (datum < ptr->max2)
            ptr->max2 = datum;
      return ptr->count;
}

#else /* C++ */

size_t Stat::Add(double datum)
{
      total   += datum;
      total2  += (datum * datum);
```

```
101 |       recip   += 1.0 / datum;
102 |       product *= datum;
103 |       ++(count);
104 |       if (datum < min1)
105 |       {
106 |             oldmin = min1;
107 |             min1 = datum;
108 |       }
109 |       else if (datum < min2)
110 |             min2 = datum;
111 |       if (datum > max1)
112 |       {
113 |             oldmax = max1;
114 |             max1 = datum;
115 |       }
116 |       else if (datum < max2)
117 |             max2 = datum;
118 |       return count;
119 | }
120 |
121 | #endif
122 |
123 | /*
124 | **  Delete a value from a statistical analysis
125 | */
126 |
127 | #if !(__cplusplus)
128 |
129 | size_t stat_delete(double datum, Stat_T *ptr)
130 | {
131 |       ptr->total   -= datum;
132 |       ptr->total2  -= (datum * datum);
133 |       ptr->recip   -= 1.0 / datum;
134 |       ptr->product /= datum;
135 |       --(ptr->count);
136 |       if (datum == ptr->min1)
137 |             ptr->min1 = (DBL_MAX == ptr->oldmin) ? ptr->min2 : ptr->oldmin;
138 |       if (datum == ptr->max1)
139 |             ptr->max1 = (DBL_MIN == ptr->oldmax) ? ptr->max2 : ptr->oldmax;
140 |       return ptr->count;
141 | }
142 |
143 | #else /* C++ */
144 |
145 | size_t Stat::Delete(double datum)
146 | {
147 |       total   -= datum;
148 |       total2  -= (datum * datum);
149 |       recip   -= 1.0 / datum;
150 |       product /= datum;
151 |       --(count);
152 |       if (datum == min1)
153 |             min1 = (DBL_MAX == oldmin) ? min2 : oldmin;
154 |       if (datum == max1)
155 |             max1 = (DBL_MIN == oldmax) ? max2 : oldmax;
156 |       return count;
157 | }
158 |
159 | #endif
160 |
161 | /*
162 | **  "Olympic filter" - toss out the high and low data
163 | */
164 |
165 | #if !(__cplusplus)
166 |
167 | Boolean_T stat_olympic(Stat_T *ptr)
168 | {
169 |       if (ptr->count < 3)
170 |             return Error_;
171 |       stat_delete(ptr->min1, ptr);
172 |       stat_delete(ptr->max1, ptr);
173 |       return Success_;
174 | }
175 |
176 | #else /* C++ */
177 |
178 | Boolean_T Stat::Olympic()
179 | {
180 |       if (count < 3)
181 |             return Error_;
182 |       Delete(min1);
183 |       Delete(max1);
184 |       return Success_;
185 | }
186 |
187 | #endif
188 |
189 | /*
190 | **  Return the minimum datum
191 | */
192 |
193 | #if !(__cplusplus)
194 |
195 | double stat_min(Stat_T *ptr)
196 | {
197 |       return ptr->min1;
198 | }
199 |
200 | #else /* C++ */
```

```
201 |
202 | inline double Stat::Min()
203 | {
204 |       return min1;
205 | }
206 |
207 | #endif
208 |
209 | /*
210 | **  Return the maximum datum
211 | */
212 |
213 | #if !(__cplusplus)
214 |
215 | double stat_max(Stat_T *ptr)
216 | {
217 |       return ptr->max1;
218 | }
219 |
220 | #else /* C++ */
221 |
222 | inline double Stat::Max()
223 | {
224 |       return max1;
225 | }
226 |
227 | #endif
228 |
229 | /*
230 | **  Return the error (%) for the minimum datum
231 | */
232 |
233 | #if !(__cplusplus)
234 |
235 | double stat_minerror(Stat_T *ptr)
236 | {
237 |       double Mean = stat_mean(ptr);
238 |
239 |       return 100.0 * ((ptr->min1 - Mean) / Mean) ;
240 | }
241 |
242 | #else /* C++ */
243 |
244 | double Stat::Minerror()
245 | {
246 |       double mean_ = Mean();
247 |
248 |       return 100.0 * ((min1 - mean_) / mean_) ;
249 | }
250 |
251 | #endif
252 |
253 | /*
254 | **  Return the error (%) for the maximum datum
255 | */
256 |
257 | #if !(__cplusplus)
258 |
259 | double stat_maxerror(Stat_T *ptr)
260 | {
261 |       double Mean = stat_mean(ptr);
262 |
263 |       return 100.0 * ((ptr->max1 - Mean) / Mean) ;
264 | }
265 |
266 | #else /* C++ */
267 |
268 | double Stat::Maxerror()
269 | {
270 |       double mean_ = Mean();
271 |
272 |       return 100.0 * ((max1 - mean_) / mean_) ;
273 | }
274 |
275 | #endif
276 |
277 | /*
278 | **  Compute the arithmetic mean
279 | */
280 |
281 | #if !(__cplusplus)
282 |
283 | double stat_mean(Stat_T *ptr)
284 | {
285 |       if (ptr->count)
286 |             return (ptr->total / ptr->count);
287 |       else  return 0.0;
288 | }
289 |
290 | #else /* C++ */
291 |
292 | double Stat::Mean()
293 | {
294 |       if (count)
295 |             return (total / count);
296 |       else  return 0.0;
297 | }
298 |
299 | #endif
300 |
```

```
301 | /*
302 | **  Compute the geometric mean
303 | */
304 |
305 | #if !(__cplusplus)
306 |
307 | double stat_gmean(Stat_T *ptr)
308 | {
309 |       if (ptr->count)
310 |             return pow(ptr->product, 1.0 / ptr->count);
311 |       else  return 0.0;
312 | }
313 |
314 | #else /* C++ */
315 |
316 | double Stat::Gmean()
317 | {
318 |       if (count)
319 |             return pow(product, 1.0 / count);
320 |       else  return 0.0;
321 | }
322 |
323 | #endif
324 |
325 | /*
326 | **  Compute the harmonic mean
327 | */
328 |
329 | #if !(__cplusplus)
330 |
331 | double stat_hmean(Stat_T *ptr)
332 | {
333 |       if (ptr->count)
334 |             return ptr->count / ptr->recip;
335 |       else  return 0.0;
336 | }
337 |
338 | #else /* C++ */
339 |
340 | double Stat::Hmean()
341 | {
342 |       if (count)
343 |             return count / recip;
344 |       else  return 0.0;
345 | }
346 |
347 | #endif
348 |
349 | /*
350 | **  Compute the standard deviation of the population
351 | */
352 |
353 | #if !(__cplusplus)
354 |
355 | double stat_stddevP(Stat_T *ptr)
356 | {
357 |       double tmp;
358 |
359 |       tmp  = ptr->total / ptr->count;
360 |       tmp *= tmp;
361 |       return sqrt((ptr->total2 / ptr->count) - tmp);
362 | }
363 |
364 | #else /* C++ */
365 |
366 | double Stat::StddevP()
367 | {
368 |       double tmp;
369 |
370 |       tmp  = total / count;
371 |       tmp *= tmp;
372 |       return sqrt((total2 / count) - tmp);
373 | }
374 |
375 | #endif
376 |
377 | /*
378 | **  Compute the standard deviation of the sampled data
379 | */
380 |
381 | #if !(__cplusplus)
382 |
383 | double stat_stddevS(Stat_T *ptr)
384 | {
385 |       double mean, total, total2, N, tmp;
386 |
387 |       if (ptr->count < 2)
388 |             return 0.0;
389 |
390 |       mean   = ptr->total / ptr->count;
391 |       total  = ptr->total - mean;
392 |       total2 = ptr->total2 - (mean * mean);
393 |       N      = ptr->count - 1;
394 |       tmp    = total / N;
395 |       tmp   *= tmp;
396 |       return sqrt((total2 / N) - tmp);
397 | }
398 |
399 | #else /* C++ */
400 |
```

```
401 | double Stat::StddevS()
402 | {
403 |       double Mean, Total, Total2, N, tmp;
404 |
405 |       if (count < 2)
406 |             return 0.0;
407 |
408 |       Mean   = total / count;
409 |       Total  = total - Mean;
410 |       Total2 = total2 - (Mean * Mean);
411 |       N      = count - 1;
412 |       tmp    = Total / N;
413 |       tmp   *= tmp;
414 |       return sqrt((Total2 / N) - tmp);
415 | }
416 |
417 | #endif
418 |
419 | /*
420 | **  Compute the variance
421 | */
422 |
423 | #if !(__cplusplus)
424 |
425 | double stat_var(Stat_T *ptr)
426 | {
427 |       return stat_stddevS(ptr) * stat_stddevS(ptr);
428 | }
429 |
430 | #else /* C++ */
431 |
432 | inline double Stat::Var()
433 | {
434 |       return StddevS() * StddevS();
435 | }
436 |
437 | #endif
438 |
439 | /*
440 | **  Compute the coefficient of variation (percentage) for the population
441 | */
442 |
443 | #if !(__cplusplus)
444 |
445 | double stat_varcoeffP(Stat_T *ptr)
446 | {
447 |       return (stat_stddevP(ptr) / stat_mean(ptr)) * 100.0;
448 | }
449 |
450 | #else /* C++ */
451 |
452 | double Stat::VarcoeffP()
453 | {
454 |       return (StddevP() / Mean()) * 100.0;
455 | }
456 |
457 | #endif
458 |
459 | /*
460 | **  Compute the coefficient of variation (percentage) for the sample
461 | */
462 |
463 | #if !(__cplusplus)
464 |
465 | double stat_varcoeffS(Stat_T *ptr)
466 | {
467 |       return (stat_stddevS(ptr) / stat_mean(ptr)) * 100.0;
468 | }
469 |
470 | #else /* C++ */
471 |
472 | double Stat::VarcoeffS()
473 | {
474 |       return (StddevS() / Mean()) * 100.0;
475 | }
476 |
477 | #endif
478 |
479 |
480 | /*
481 | **  Test harness begins here. Define TEST macro to build stand-alone.
482 | */
483 |
484 | #ifdef TEST
485 |
486 | #include <stdlib.h>
487 |
488 | #if !(__cplusplus)
489 |
490 | #include <stdio.h>
491 |
492 | int main(int argc, char *argv[])
493 | {
494 |       Stat_T data;
495 |
496 |       stat_init(&data);
497 |       while (--argc)
498 |       {
499 |             double ftmp;
500 |
```

```
501 |             ftmp = atof(*(++argv));
502 |             stat_add(ftmp, &data);
503 |       }
504 |       puts("\nBefore \"Olympic\" filtering\n");
505 |       printf("Minimum datum             = %g\n", stat_min(&data));
506 |       printf("Maximum datum             = %g\n", stat_max(&data));
507 |       printf("Number of samples         = %d\n", stat_count(&data));
508 |       printf("Arithmetic mean           = %g\n", stat_mean(&data));
509 |       printf("Geometric mean            = %g\n", stat_gmean(&data));
510 |       printf("Harmonic mean             = %g\n", stat_hmean(&data));
511 |       printf("Standard deviation (N)    = %g\n", stat_stddevP(&data));
512 |       printf("Standard deviation (N-1)  = %g\n", stat_stddevS(&data));
513 |       printf("Variance                  = %g\n", stat_var(&data));
514 |       printf("Population coeff. of var. = %g%%\n", stat_varcoeffP(&data));
515 |       printf("Sample coeff. of var.     = %g%%\n", stat_varcoeffS(&data));
516 |
517 |       puts("\nAfter \"Olympic\" filtering\n");
518 |       printf("stat_olympic() returned %s\n", stat_olympic(&data) ?
519 |             "ERROR" : "SUCCESS");
520 |       printf("Minimum datum             = %g\n", stat_min(&data));
521 |       printf("Maximum datum             = %g\n", stat_max(&data));
522 |       printf("Number of samples         = %d\n", stat_count(&data));
523 |       printf("Arithmetic mean           = %g\n", stat_mean(&data));
524 |       printf("Geometric mean            = %g\n", stat_gmean(&data));
525 |       printf("Harmonic mean             = %g\n", stat_hmean(&data));
526 |       printf("Standard deviation (N)    = %g\n", stat_stddevP(&data));
527 |       printf("Standard deviation (N-1)  = %g\n", stat_stddevS(&data));
528 |       printf("Variance                  = %g\n", stat_var(&data));
529 |       printf("Population coeff. of var. = %g%%\n", stat_varcoeffP(&data));
530 |       printf("Sample coeff. of var.     = %g%%\n", stat_varcoeffS(&data));
531 |
532 |       return EXIT_SUCCESS;
533 | }
534 |
535 | #else /* C++ */
536 |
537 | #include <iostreams.h>
538 |
539 | int main(int argc, char *argv[])
540 | {
541 |       class Stat data;
542 |       char *str;
543 |
544 |       while (--argc)
545 |       {
546 |             double ftmp;
547 |
548 |             ftmp = atof(*(++argv));
549 |             data.Add(ftmp);
550 |       }
551 |       cout << endl << "Before \"Olympic\" filtering\n" << endl << endl;
552 |       cout << "Minimum datum             = " << data.Min()       << endl;
553 |       cout << "Maximum datum             = " << data.Max()       << endl;
554 |       cout << "Number of samples         = " << data.Count()     << endl;
555 |       cout << "Arithmetic mean           = " << data.Mean()      << endl;
556 |       cout << "Geometric mean            = " << data.Gmean()     << endl;
557 |       cout << "Harmonic mean             = " << data.Hmean()     << endl;
558 |       cout << "Standard deviation (N)    = " << data.StddevP()   << endl;
559 |       cout << "Standard deviation (N-1)  = " << data.StddevS()   << endl;
560 |       cout << "Variance                  = " << data.Var()       << endl;
561 |       cout << "Population coeff. of var. = " << data.VarcoeffP() << "%" << endl;
562 |       cout << "Sample coeff. of var.     = " << data.VarcoeffS() << "%" << endl;
563 |
564 |       if (Success_ == data.Olympic())
565 |             str = "SUCCESS";
566 |       else  str = "ERROR";
567 |
568 |       cout << endl << "After \"Olympic\" filtering" << endl << endl;
569 |       cout << "data.Olympic() returned " << str << endl;
570 |       cout << "Minimum datum             = " << data.Min()       << endl;
571 |       cout << "Maximum datum             = " << data.Max()       << endl;
572 |       cout << "Number of samples         = " << data.Count()     << endl;
573 |       cout << "Arithmetic mean           = " << data.Mean()      << endl;
574 |       cout << "Geometric mean            = " << data.Gmean()     << endl;
575 |       cout << "Harmonic mean             = " << data.Hmean()     << endl;
576 |       cout << "Standard deviation (N)    = " << data.StddevP()   << endl;
577 |       cout << "Standard deviation (N-1)  = " << data.StddevS()   << endl;
578 |       cout << "Variance                  = " << data.Var()       << endl;
579 |       cout << "Population coeff. of var. = " << data.VarcoeffP() << "%" << endl;
580 |       cout << "Sample coeff. of var.     = " << data.VarcoeffS() << "%" << endl;
581 |
582 |       return EXIT_SUCCESS;
583 | }
584 |
585 | #endif /* C/C++ */
586 |
587 | #endif
```

```
TEXT STATISTICS

 12431 characters
   587 lines

LEXICAL STATISTICS

    39 comments [std-C]
     0 comments [C++]
    62 preprocessor instructions
     0 constants [character]
    59 constants [string]
    38 constants [numeric]

SYMBOL TABLE

    Add             97     549
    Boolean_T      167     178
    Count           58     554     572
    DBL_MAX         26      39     137     153
    DBL_MIN         27      40     139     155
    Delete         145     182     183
    EXIT_SUCCESS           532     582
    Error_         170     181
    Gmean          316     556     574
    Hmean          340     557     575
    Max            222     553     571
    Maxerror       268
    Mean           237     239+    246     261     263+    270     292     403     408     409     410+
           454     474     555     573
    Min            202     552     570
    Minerror       244
    N      385     393     394     396     403     411     412     414
    Olympic        178     564
    Stat            32+     58      97     145     178     202     222     244     268     292     316
           340     366     401     432     452     472     541
    Stat_T          19      51      71     129     167     195     215     235     259     283     307
           331     355     383     425     445     465     494
    StddevP        366     454     558     576
    StddevS        401     434+    474     559     577
    Success_       173     184     564
    TEST           484
    Total          403     409     412
    Total2         403     410     414
    Var            432     560     578
    VarcoeffP      452     561     579
    VarcoeffS      472     562     580
    __cplusplus             17      49      69     127     165     193     213     233     257     281
           305     329     353     381     423     443     463     488
    argc           492     497     539     544
    argv           492     501     539     548
    atof           501     548
    count           21      34      53      60      77      92     103     118     135     140     151
           156     169     180     285     286     294     295     309     310     318     319     333
           334     342     343     359     361     370     372     387     390     393     405     408
           411
    cout           551     552     553     554     555     556     557     558     559     560     561
           562     568     569     570     571     572     573     574     575     576     577     578
           579     580
    data           494     496     502     505     506     507     508     509     510     511     512
           513     514     515     518     520     521     522     523     524     525     526     527
           528     529     530     541     549     552     553     554     555     556     557     558
           559     560     561     562     564     570     571     572     573     574     575     576
           577     578     579     580
    datum           71      73      74+     75      76      78      81      83      84      85      88
            90      91      97      99     100+    101     102     104     107     109     110     111
           114     116     117     129     131     132+    133     134     136     138     145     147
           148+    149     150     152     154
    endl           551+    552     553     554     555     556     557     558     559     560     561
           562     568+    569     570     571     572     573     574     575     576     577     578
           579     580
    ftmp           499     501     502     546     548     549
    h        9      10     486     490     537
    iostreams      537
    main           492     539
    max1            27      40      85      87      88     111     113     114     138     139     154
           155     172     183     217     224     263     272
    max2            27      40      90      91     116     117     139     155
    mean           385     390     391     392+
    mean_          246     248+    270     272+
    min1            26      39      78      80      81     104     106     107     136     137     152
           153     171     182     197     204     239     248
    min2            26      39      83      84     109     110     137     153
    oldmax          27      40      87     113     139+    155+
    oldmin          26      39      80     106     137+    153+
    pow            310     319
    printf         505     506     507     508     509     510     511     512     513     514     515
           518     520     521     522     523     524     525     526     527     528     529     530
    product         25      38      76     102     134     150     310     319
    ptr             19      21      22      23      24      25      26+     27+     51      53      71
            73      74      75      76      77      78      80+     81      83      84      85      87+
            88      90      91      92     129     131     132     133     134     135     136     137+
           138     139+    140     167     169     171+    172+    195     197     215     217     235
           237     239     259     261     263     283     285     286+    307     309     310+    331
           333     334+    355     359+    361+    383     387     390+    391     392     393     425
           427+    445     447+    465     467+
    puts           504     517
    recip           24      37      75     101     133     149     334     343
    size_t          51      58      71      97     129     145
    sqrt           361     372     396     414
```

```
stat_add         71     502
stat_count                51     507     522
stat_delete              129     171     172
stat_gmean               307     509     524
stat_hmean               331     510     525
stat_init         19     496
stat_max         215     506     521
stat_maxerror            259
stat_mean        237     261     283     447     467     508     523
stat_min         195     505     520
stat_minerror            235
stat_olympic             167     518
stat_stddevP             355     447     511     526
stat_stddevS             383     427+    467     512     527
stat_var         425     513     528
stat_varcoeffP           445     514     529
stat_varcoeffS           465     515     530
stddef             9
stdio            490
stdlib           486
str              542     565     566     569
tmp              357     359     360+    361     368     370     371+    372     385     394     395+
        396      403     412     413+    414
total             22      35      73      99     131     147     286     295     359     370     385
        390      391+    394     408     409
total2            23      36      74     100     132     148     361     372     385     392+    396
        410
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MATHSTAT.H - Header file for statistical analysis in C and C++
**
**  Public domain by Bob Stout
*/

#include <stddef.h>
#include <math.h>
#include "sniptype.h"

#if !(__cplusplus)

/*
**  Structure to hold statistical analysis data
**
**  total   = Total of all data added
**  total2  = Total of squares of all data added
**  product = Product of all data added
**  recip   = Total of the reciprocals of all data added
**  count   = Number of data elements under analysis
**  min1    = Minimum datum to date
**  min2    = Next to lowest datum to date
**  oldmin  = Previous value of min1
**  max1    = Maximum datum to date
**  max2    = Next to highest datum to date
**  oldmax  = Previous value of max1
**
**  Notes: min2, oldmin, max2, and oldmax are used when deleting data.
**         If the datum to be deleted is either the highest or lowest to
**         date, it must be replaced with either the previous min/max
**         value (oldmin/oldmax) or the penultimate min/max value (min2/max2).
**         Replacement with the penultimate value is indicated when the
**         previous min/max value was either DBL_MAX or DBL_MIN,
**         respectively, indicating the initialized condition.
**
**       The C++ Stat class works identically to the functions used with
**       the Stat_T structure in C. All members of Stat_T are present as
**       private data objects in the Stat class. All functions associated
**       with Stat_T are duplicated as member functions of class Stat.
**
**         Note that the data are not saved, therefore it is impossible to
**         provide functions such as one to return the median of a data set.
*/

typedef struct {
      size_t count;
      double total;
      double total2;
      double product;
      double recip;
      double min1, min2, oldmin;
      double max1, max2, oldmax;
} Stat_T;

void        stat_init(Stat_T *ptr);
size_t      stat_count(Stat_T *ptr);
size_t      stat_add(double datum, Stat_T *ptr);
size_t      stat_delete(double datum, Stat_T *ptr);
Boolean_T   stat_olympic(Stat_T *ptr);
double      stat_min(Stat_T *ptr);
double      stat_max(Stat_T *ptr);
double      stat_minerror(Stat_T *ptr);
double      stat_maxerror(Stat_T *ptr);
double      stat_mean(Stat_T *ptr);
double      stat_gmean(Stat_T *ptr);
double      stat_hmean(Stat_T *ptr);
double      stat_stddevP(Stat_T *ptr);
double      stat_stddevS(Stat_T *ptr);
double      stat_var(Stat_T *ptr);
double      stat_varcoeffP(Stat_T *ptr);
double      stat_varcoeffS(Stat_T *ptr);

#else /* is C++ */

class Stat {
      private:
            double            total;
            double            total2;
            double            product;
            double            recip;
            size_t            count;
            double            min1, min2, oldmin;
            double            max1, max2, oldmax;
      public:
            inline            Stat();
            inline size_t     Count();
            size_t            Add(double datum);
            size_t            Delete(double datum);
            Boolean_T         Olympic();
            inline double     Min();
            inline double     Max();
            double            Minerror();
            double            Maxerror();
            double            Mean();
            double            Gmean();
            double            Hmean();
            double            StddevP();
            double            StddevS();
```

```
 101 |             inline double     Var();
 102 |             double            VarcoeffP();
 103 |             double            VarcoeffS();
 104 | };
 105 |
 106 | #endif
```

TEXT STATISTICS

```
  3731 characters
   106 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Add             89
    Boolean_T       61      91
    Count           88
    Delete          90
    Gmean           97
    Hmean           98
    Max             93
    Maxerror        95
    Mean            96
    Min             92
    Minerror        94
    Olympic         91
    Stat            77      87
    Stat_T          55      57      58      59      60      61      62      63      64      65      66
          67        68      69      70      71      72      73
    StddevP         99
    StddevS        100
    Var            101
    VarcoeffP      102
    VarcoeffS      103
    __cplusplus             13
    count           48      83
    datum           59      60      89      90
    h         9     10
    math            10
    max1            54      85
    max2            54      85
    min1            53      84
    min2            53      84
    oldmax          54      85
    oldmin          53      84
    product         51      81
    ptr             57      58      59      60      61      62      63      64      65      66      67
          68        69      70      71      72      73
    recip           52      82
    size_t          48      58      59      60      83      88      89      90
    stat_add        59
    stat_count              58
    stat_delete             60
    stat_gmean              67
    stat_hmean              68
    stat_init       57
    stat_max        63
    stat_maxerror           65
    stat_mean       66
    stat_min        62
    stat_minerror           64
    stat_olympic            61
    stat_stddevP            69
    stat_stddevS            70
    stat_var        71
    stat_varcoeffP          72
    stat_varcoeffS          73
    stddef           9
    total           49      79
    total2          50      80
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** maxline.c - returns the length of the longest line in text file
  5 | **
  6 | ** Released to Public Domain by Phi Nguyen
  7 | **
  8 | ** 03.22.96 - First release
  9 | ** 03.26.96 - Chad Wallace, for use with FILECAT.C
 10 | **              Uses an array of pointers to strings (ala argv)
 11 | **              instead of a file.
 12 | */
 13 |
 14 | #include <stdio.h>
 15 | #include <string.h>
 16 |
 17 | unsigned int max_line(char ** str_array)
 18 | {
 19 |       unsigned cur_len, max = 0;
 20 |
 21 |       while (*str_array != NULL)
 22 |       {
 23 |             cur_len = strlen(*str_array);
 24 |             if (cur_len > max)
 25 |                   max = cur_len;
 26 |
 27 |             str_array++;
 28 |       }
 29 |
 30 |       return max;
 31 | }
 32 |
 33 | #ifdef TEST
 34 |
 35 | /*
 36 | ** Will return the length of the largest parameter given on the
 37 | ** command line, including argv[0], the program's .EXE file.
 38 | */
 39 |
 40 | main(int argc, char **argv)
 41 | {
 42 |       printf("%i\n", max_line(argv));
 43 |
 44 |       return 0;
 45 | }
 46 |
 47 | #endif
 48 |
```

TEXT STATISTICS

```
   952 characters
    48 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    NULL            21
    TEST            33
    argc            40
    argv            40      42
    cur_len         19      23      24      25
    h       14      15
    main            40
    max             19      24      25      30
    max_line        17      42
    printf          42
    stdio           14
    str_array       17      21      23      27
    string          15
    strlen          23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
   This program makes 10x10 mazes and prints them on the screen.  No
   promise of portability is made, but it does seem to work on NS GNX
   C.

   Public Domain by Jonathan Guthrie.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define UP  1
#define DN  2
#define LT  4
#define RT  8

int  addelem(int, int [12][12], int *, int, int);
void openwall(int [12][12], int, int);
void writemaze(int [12][12]);

main()
{
      int i, j, base;
      int search[150], array[12][12];

      for(i=1 ; i<11 ; ++i)
      {
            array[i][0] = -1;
            array[i][11] = -1;
            array[0][i] = -1;
            array[11][i] = -1;
            for(j=1 ; j<11 ; ++j)
                  array[i][j] = 0;
      }

      srand((int)time(NULL));
      i = rand() % 10 + 1;
      j = rand() % 10 + 1;
      base = addelem(0, array, search, i, j);
      array[i][j] = RT + RT;      /* Not a valid value */
      while(0 < base)
      {
            i = rand() % base;
            j = search[i];
            search[i] = search[--base];
            i = j % 100;
            j /= 100;
            openwall(array, i, j);
            base = addelem(base, array, search, i, j);
      }

      writemaze(array);
      return 0;
}


int addelem(int base, int maze[12][12], int *search, int row, int col)
{
      if(0 == maze[row-1][col])
      {
            search[base++] = row + col * 100 - 1;
            maze[row-1][col] = -DN;
      }
      else if(0 > maze[row-1][col])
            maze[row-1][col] -= DN;

      if(0 == maze[row+1][col])
      {
            search[base++] = row + col * 100 + 1;
            maze[row+1][col] = -UP;
      }
      else if(0 > maze[row+1][col])
            maze[row+1][col] -= UP;

      if(0 == maze[row][col-1])
      {
            search[base++] = row + col * 100 - 100;
            maze[row][col-1] = -RT;
      }
      else if(0 > maze[row][col-1])
            maze[row][col-1] -= RT;

      if(0 == maze[row][col+1])
      {
            search[base++] = row + col * 100 + 100;
            maze[row][col+1] = -LT;
      }
      else if(0 > maze[row][col+1])
            maze[row][col+1] -= LT;

      return base;
}


void openwall(int maze[12][12], int row, int col)
{
      int directions, max, direction, temprow, tempcol, temp, back;
```

```
101 |
102 |       directions = -maze[row][col];
103 |
104 |       max = 0;
105 |       if(directions & UP)
106 |       {
107 |             temp = rand();
108 |             if(temp > max)
109 |             {
110 |                   max = temp;
111 |                   direction = UP;
112 |                   back = DN;
113 |                   temprow = row - 1;
114 |                   tempcol = col;
115 |             }
116 |       }
117 |
118 |       if(directions & DN)
119 |       {
120 |             temp = rand();
121 |             if(temp > max)
122 |             {
123 |                   max = temp;
124 |                   direction = DN;
125 |                   back = UP;
126 |                   temprow = row + 1;
127 |                   tempcol = col;
128 |             }
129 |       }
130 |
131 |       if(directions & LT)
132 |       {
133 |             temp = rand();
134 |             if(temp > max)
135 |             {
136 |                   max = temp;
137 |                   direction = LT;
138 |                   back = RT;
139 |                   temprow = row;
140 |                   tempcol = col - 1;
141 |             }
142 |       }
143 |
144 |       if(directions & RT)
145 |       {
146 |             temp = rand();
147 |             if(temp > max)
148 |             {
149 |                   max = temp;
150 |                   direction = RT;
151 |                   back = LT;
152 |                   temprow = row;
153 |                   tempcol = col + 1;
154 |             }
155 |       }
156 |
157 |       maze[row][col] = direction;
158 |       maze[temprow][tempcol] += back;
159 | }
160 |
161 | void writemaze(int maze[12][12])
162 | {
163 |       int i, j;
164 |
165 |       puts("**********************");
166 |       for(i=1 ; i<11 ; ++i)
167 |       {
168 |             putchar('*');
169 |             for(j=1 ; j<11 ; ++j)
170 |             {
171 |                   putchar(' ');
172 |                   if(maze[i][j] & RT)
173 |                         putchar(' ');
174 |                   else  putchar('*');
175 |             }
176 |             putchar('\n');
177 |             for(j=1 ; j<11 ; ++j)
178 |             {
179 |                   putchar('*');
180 |                   if(maze[i][j] & DN)
181 |                         putchar(' ');
182 |                   else  putchar('*');
183 |             }
184 |             puts("*");
185 |       }
186 | }
```

TEXT STATISTICS

```
  4436 characters
   186 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     8 constants [character]
     2 constants [string]
    84 constants [numeric]
```

SYMBOL TABLE

```
    DN              16      65      68     112     118     124     180
    LT              17      89      92     131     137     151
    NULL            39
    RT              18      43+     81      84     138     144     150     172
    UP              15      73      76     105     111     125
    addelem         20      42      52      60
    array           27      31      32      33      34      36      42      43      51      52      55
    back           100     112     125     138     151     158
    base            26      42      44      46      48      52+     60      64      72      80      88
           94
    col             60      62      64      65      67      68      70      72      73      75      76
           78      80      81      83      84      86      88      89      91      92      98     102
          114     127     140     153     157
    direction      100     111     124     137     150     157
    directions             100     102     105     118     131     144
    h       11      12      13
    i       26      29+     31      32      33      34      36      40      42      43      46      47
           48      49      51      52     163     166+    172     180
    j       26      35+     36      41      42      43      47      49      50      51      52     163
          169+    172     177+    180
    main            24
    max            100     104     108     110     121     123     134     136     147     149
    maze            60      62      65      67      68      70      73      75      76      78      81
           83      84      86      89      91      92      98     102     157     158     161     172
          180
    openwall        21      51      98
    putchar        168     171     173     174     176     179     181     182
    puts           165     184
    rand            40      41      46     107     120     133     146
    row             60      62      64      65      67      68      70      72      73      75      76
           78      80      81      83      84      86      88      89      91      92      98     102
          113     126     139     152     157
    search          27      42      47      48+     52      60      64      72      80      88
    srand           39
    stdio           11
    stdlib          12
    temp           100     107     108     110     120     121     123     133     134     136     146
          147     149
    tempcol        100     114     127     140     153     158
    temprow        100     113     126     139     152     158
    time            13      39
    writemaze       22      55     161
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | int a[1817];main(z,p,q,r){for(p=80;q+p-80;p-=2*a[p])for(z=9;z--;)q=3&(r=time(0)+r*57)/7,q=q?q-1?q-2?1-
     |  p%79?-1:0:p%79-77?1:0:p<1659?79:0:p>158?-79:0,q?!a[p+q*2]?a[p+=a[p+=q]=q]=q:0:0;for(;q++-1817;)printf
     |  (q%79?"%c":"%c\n"," #"[!a[q-1]]);}
```

TEXT STATISTICS

```
   283 characters
     3 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     3 constants [string]
    30 constants [numeric]
```

SYMBOL TABLE

```
    a        3+
    main             3
    p        3+
    printf           3
    q        3+
    r        3+
    time             3
    z        3+
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | char*M,A,Z,E=40,J[40],T[40];main(C){for(*J=A=scanf(M="%d",&C);
   4 | --            E;            J[            E]            =T
   5 | [E   ]=  E)   printf("._");  for(;(A-=Z=!Z)  ||  (printf("\n|"
   6 | )    ,   A    =             39            ,C            --
   7 | )    ;   Z    ||    printf   (M   ))M[Z]=Z[A-(E   =A[J-Z])&&!C
   8 | &    A   ==            T[                                  A]
   9 | |6<<11<rand()|||!C&!Z?J[T[E]=T[A]]=E,J[T[A]=A-Z]=A,"_.":" |"];}
```

TEXT STATISTICS

```
   492 characters
     9 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     5 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    A        3+      5       6       7+      8+      9+
    C        3+      6       7       9
    E        3       4+      5+      7       9+
    J        3+      4       7       9+
    M        3+      7+
    T        3       4       8       9+
    Z        3       5+      7+      9+
    main             3
    printf           5+      7
    rand             9
    scanf            3
```

```
/* +++Date last modified: 05-Jul-1997 */

#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <ctype.h>

#if defined(__TURBOC__) || defined(__POWERC) || defined(__ZTC__)
 #if defined(__ZTC__)
  #include <dos.h>
 #else
  #include <alloc.h>
 #endif
#else                                           /* MSC, Watcom    */
 #include <malloc.h>
#endif

#include "extkword.h"
#include "mk_fp.h"

struct EnvRec {
              unsigned EnvSeg;  /*Segment of the environment*/
              unsigned EnvLen;  /*Usable length of the environment*/
              } EnvRec;

struct EnvRec *MasterEnv(void)
{
      unsigned owner;
      unsigned mcb;
      unsigned eseg;
      static struct EnvRec env;

      env.EnvSeg = env.EnvLen = 0;
      owner = * ((unsigned FAR *) MK_FP(0, (2+4*0x2e)));

      /* int 0x2e points to command.com */

      mcb = owner -1;

      /*Mcb points to memory control block for COMMAND */

      if ( (*((char FAR *) MK_FP(mcb, 0)) != 'M') &&
           (*((unsigned FAR *) MK_FP(mcb, 1)) != owner) )
                return (struct EnvRec *) 0;

      eseg = *((unsigned FAR *) MK_FP(owner, 0x2c));

      /* Read segment of environment from PSP of COMMAND} */
      /* Earlier versions of DOS don't store environment segment there */

      if ( !eseg )
      {

            /* Master environment is next block past COMMAND */

            mcb = owner + *((unsigned FAR *) MK_FP(mcb, 3));
            if ( (*((char FAR *) MK_FP(mcb, 0)) != 'M') &&
                 (*((unsigned FAR *) MK_FP(mcb, 1)) != owner) )
                      return (struct EnvRec *) 0;
            eseg = mcb + 1;
      }
      else  mcb = eseg-1;

      /* Return segment and length of environment */

      env.EnvSeg = eseg;
      env.EnvLen = *((unsigned FAR *) MK_FP(mcb, 3)) << 4 ;
      return &env;
}

/*
** Then a function to find the string to be replaced.   This one'll
** return a pointer to the string, or a pointer to the first (of 2)
** NUL byte at the end of the environment.
*/

char FAR *SearchEnv( char FAR *eptr, char *search )
{
      char FAR *e;
      char *s;
      while ( *eptr )
      {
            for ( s=search, e=eptr; *e && *s && (*e == *s); e++, s++ )
                  ;  /* NULL STATEMENT */
            if ( !*s )
                  break;
            while ( *eptr )
                  eptr++;           /* position to the NUL byte */
            eptr++;                     /* next string */
      }
      return eptr;
}

/*
** Now, the function to replace, add or delete.  If a value is not
** given, the string is deleted.
*/

int SetEnvStr( struct EnvRec *env, char *search, char *value )
{
```

```
101 |       /* -Set environment string, returning true if successful */
102 |
103 |       char FAR *envptr;
104 |       register char FAR *p;
105 |       char *s;
106 |       int newlen;
107 |       int oldlen;
108 |       int i;
109 |
110 |       if ( !env->EnvSeg || !search )
111 |             return 0;
112 |
113 |       /* get some memory for complete environment string */
114 |
115 |       newlen = strlen(search) + sizeof((char) '\0') + strlen(value) + 2;
116 |       if ( (s = (char *) malloc( newlen)) == NULL )
117 |             return 0;
118 |       for ( i = 0; *search; search++, i++ )
119 |             s[i] = *search;
120 |       s[i++] = '=';
121 |       s[i] = '\0';
122 |       envptr = SearchEnv((char FAR *) MK_FP(env->EnvSeg, 0), s );
123 |       if ( *envptr )
124 |       {
125 |             for ( p = envptr, oldlen = 0; *p; oldlen++, p++ )
126 |                   ;     /* can't use strlen() because of far pointer */
127 |       }               /* will set p to point to terminating NUL */
128 |
129 |       if ( *value && (newlen > (int)env->EnvLen) ) /* not a deletion */
130 |       {
131 |             free( s );
132 |             return 0;                          /* won't fit */
133 |       }
134 |
135 |       if ( *envptr )                           /* shift it down */
136 |       {
137 |             for ( ++p; (*p || *(p+1)); envptr++, p++ )
138 |                   *envptr = *p;
139 |             *envptr++ = '\0';
140 |             *envptr = '\0';
141 |       }
142 |       if ( *value )                            /* append it */
143 |       {
144 |             strcat(s, value);
145 |             while ( *s )
146 |                   *(envptr++) = *s++;
147 |             *envptr++ = '\0';
148 |             *envptr = '\0';
149 |       }
150 |       free(s);
151 |       return 1;
152 | }
153 |
154 | /*
155 | ** Ok, just to show you that I tested it :
156 | */
157 |
158 | main()
159 | {
160 |       char vn[80];
161 |       char va[80];
162 |       struct EnvRec *p = MasterEnv();
163 |
164 |       puts("enter variable name:");
165 |       gets(vn);
166 |       puts("enter value:");
167 |       gets(va);
168 |       printf("SetEnvStr returned %d\n", SetEnvStr( p, strupr(vn), va) );
169 |       return 0;
170 | }
```

TEXT STATISTICS

```
  4666 characters
   170 lines
```

LEXICAL STATISTICS

```
    24 comments [std-C]
     0 comments [C++]
    15 preprocessor instructions
     9 constants [character]
     5 constants [string]
    29 constants [numeric]
```

SYMBOL TABLE

```
    EnvLen          23      33      67     129
    EnvRec          21      24      26      31      44      59      99     162
    EnvSeg          22      33      66     110     122
    FAR             34      42      43      46      56      57      58      67      77+     79     103
          104     122
    MK_FP           34      42      43      46      56      57      58      67     122
    MasterEnv       26     162
    NULL           116
    SearchEnv       77     122
    SetEnvStr       99     168
    __POWERC         8
    __TURBOC__               8
    __ZTC__          8       9
    alloc           12
    ctype            6
    defined          8+      9
    dos              4      10
    e       79      83+
    env             31      33+     66      67      68      99     110     122     129
    envptr         103     122     123     125     135     137     138     139     140     146     147
          148
    eptr            77      81      83      87      88      89      91
    eseg            30      46      51      60      62      66
    free           131     150
    gets           165     167
    h        3       4       5       6      10      12      15
    i      108     118+    119     120     121
    main           158
    malloc          15     116
    mcb             29      38      42      43      56+     57      58      60      62      67
    newlen         106     115     116     129
    oldlen         107     125+
    owner           28      34      38      43      46      56      58
    p      104     125+    137+    138     162     168
    printf         168
    puts           164     166
    s       80      83+     85     105     116     119     120     121     122     131     144     145
          146     150
    search          77      83      99     110     115     118+    119
    stdio            3
    strcat         144
    string           5
    strlen         115+
    strupr         168
    va             161     167     168
    value           99     115     129     142     144
    vn             160     165     168
```

```
/* +++Date last modified: 05-Jul-1997 */

/*   Written by Blair Haukedal 91/09 and placed in the public domain */

/*  mdalloc - a multi dimensional array allocator
 *  mdfree  - a companion function to mdalloc for freeing storage
 *  synopsis:
 *      void *mdalloc(int ndim, int width, ...);
 *          where:  ndim:   number of array dimensions
 *                  width:  size of elements in array
 *                  variable args are dimensions of array
 *          returns: n-way indirect pointer to allocated storage
 *                   or NULL if insufficient storage
 *
 *      void mdfree(void *p, ndim);
 *          where:  p:      pointer to storage obtained by mdalloc
 *                  ndim:   number of dimensions used in mdalloc
 *
 *  example:
 *      int ***tip;
 *      tip = mdalloc(3, sizeof(int), 2, 3, 4);
 *        tip will be a triple indirect pointer to a 3 dimensional array
 *        tip[0][0][0] refers to the first int in a contiguous area of
 *                     storage that is 2*3*4*sizeof(int) bytes long
 *        tip[0][0] is the address of the first int
 *      memset can be used to initialize array elements as follows:
 *        memset(tip[0][0], 0, 2*3*4*sizeof(int));
 *      mdfree is used to free storage obtained with mdalloc:
 *        mdfree(tip, 3)
 *
 *  notes:
 *      - must be compiled with appropriate memory model
 *      - memory is allocated for each dimension for indirect pointers
 *          eg. 3x4x5 array of longs
 *              (assuming 4 byte longs, small mem model)
 *              p = mdalloc(3, sizeof(long), 3, 4, 5)           - bytes
 *                  3       pointers allocated for 1st dimension -  6
 *                  3x4     pointers allocated for 2nd dimension -  24
 *                  3x4x5   longs allocated for array elements   -  240
 *              total of 270 bytes allocated
 *      - if insufficient memory, nothing will be allocated.
 *          ie. intermediate pointer arrays that were successfully
 *              allocated will be freed.
 *      - the intent of mdalloc is to facilitate dynamic array creation,
 *        it will use more memory than statically declared arrays, and
 *        the required dereferencing will be slower than the use of
 *        statically declared arrays.
 *      - this function assumes that sizeof(char) == 1.
 */

#include <stdarg.h>
#include <stdlib.h>
#include "snparray.h"

static void **md2(int n_units, int ndim, int *dims);
static void md3(char ***tip, int n_units, int ndim, int *dims);

static int w_units;

/* mdalloc: entry point for mdalloc function described above
 *      - reduces variable arg list to fixed list with last arg
 *      represented as pointer to int (array dimensions).
 *      Calls md2 to allocate storage.
 *      Calls md3 to initialize intermediate pointers.
 *      Returns pointer.
 */

void *mdalloc(int ndim, int width, ...)
{
      va_list argp;
      int *dims, i;
      char ***tip;

      va_start(argp, width);

      /* allocate storage for variable args (dimensions) */

      dims = malloc(ndim*sizeof(int));
      if(dims == NULL)
            return NULL;

      /* initialize dimensions array for subsequent calls */

      for(i=0; i<ndim; i++)
            dims[i] = va_arg(argp,int);

      w_units = width;    /* global used by md2 and md3 */

      /* allocate required pointer and array element storage */

      tip = (char ***)md2(dims[0], ndim, &dims[1]);

      if(ndim>1 && tip)
            md3(tip, dims[0], ndim-1, &dims[1]); /* init pointers */

      free(dims);
      return tip;
}

/* mdfree:  companion function to mdalloc
```

```
 *          frees storage obtained by mdalloc
 */

void mdfree(void *tip, int ndim)
{
      if(ndim == 1)
            free(tip);
      else
      {
            mdfree(((void **)tip)[0], ndim-1);
            free(tip);
      }
}

/* md2:  allocates storage for n-way indirect pointer arrays
 *       allocates storage for requested array elements
 */

static void **md2(int n_units, int ndim, int *dims)
{
      char **tip;

      if(ndim == 1)
            /* recursed to final dimension - allocate element storage */
            tip = malloc(n_units*w_units);
      else
      {
            /* allocate pointer array for dimension n */
            tip = malloc(n_units*sizeof(char *));
            if(tip)
            {
                  /* recurse until final dimension */
                  tip[0] = (char *)md2(n_units*dims[0], ndim-1, &dims[1]);
                  if(tip[0] == NULL)
                  {
                        /* allocate error - fall back up freeing everything */
                        free(tip);
                        tip = NULL;
                  }
            }
      }
      return (void **)tip;
}

/* md3: initializes indirect pointer arrays */

static void md3(char ***tip, int n_units, int ndim, int *dims)
{
      int i;

      for(i=1; i<n_units; i++)
      {
            if(ndim == 1)
                  /* final dimension - must scale by element width */
                  tip[i] = (char **)((char *)tip[0] + i*dims[0]*w_units);
            else
                  /* intermediate dimension - scale by pointer size */
                  tip[i] = tip[0] + i*dims[0];
      }
      if(ndim > 1)
            /* not at final dimension - continue to recurse */
            md3((char ***)tip[0], n_units*dims[0], ndim-1, &dims[1]);
}
```

TEXT STATISTICS

```
  5549 characters
   163 lines
```

LEXICAL STATISTICS

```
    19 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     1 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    NULL            79      80     134     138
    argp            70      74      85
    dims            55      56      71      78      79      85      91+     94+     96     119     133+
          147     155     158     162+
    free            96     107     111     137
    h       51      52
    i       71      84+     85     149     151+    155+    158+
    malloc          78     125     129
    md2             55      91     119     133
    md3             56      94     147     162
    mdalloc         68
    mdfree         104     110
    n_units         55      56     119     125     129     133     147     151     162
    ndim            55      56      68      78      84      91      93      94     104     106     110
          119     123     133     147     153     160     162
    stdarg          51
    stdlib          52
    tip             56      72      91      93      94      97     104     107     110     111     121
          125     129     130     133     134     137     138     142     147     155+    158+    162
    va_arg          85
    va_list         70
    va_start        74
    w_units         58      87     125     155
    width           68      74      87
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MDASORT.C - Test program for mdalloc()/amalloc() demonstrating sorting
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <limits.h>
#include "snparray.h"
#include "bc_rand.h"

int compare(const void *elem1, const void *elem2);
int posn;

void usage(void)
{
      puts("Usage: MDATEST {A | M}");
      puts("where: A means to use amalloc()");
      puts("       M means to use mdalloc()");
      exit(EXIT_FAILURE);
}

main(int argc, char *argv[])
{
      int i, arg;
      char **ary;

      if (argc < 2)
            usage();
      arg = toupper(*argv[1]);
      if (arg != 'M' && arg != 'A')
            usage();
      printf("MDATEST: Using %s\n\n",
             ('M' == arg) ? "mdalloc()" : "amalloc()");

      /*
      **  Create a 5 x 40 array of char to hold 5 strings
      */

      if ('M' == arg)
      {
            if (NULL == (ary = mdalloc(sizeof(char), 5, 40)))
            {
                  puts("mdalloc() failed");
                  return EXIT_FAILURE;
            }
      }
      else
      {
            if (NULL == (ary = amalloc(sizeof(char), NULL, 2, 5, 40)))
            {
                  puts("amalloc() failed");
                  return EXIT_FAILURE;
            }
      }

      /*
      **  Fill the array with recognizable random strings
      */

      randomize();
      for (i = 0; i < 5; ++i)
      {
            posn = sprintf(ary[i], "String #%d val = ", i + 1);
            sprintf(ary[i] + posn, "%04X", random(UINT_MAX));
      }

      /*
      **  Display the unsorted array
      */

      for (i = 0; i < 5; ++i)
            printf("%d - %s\n", i, ary[i]);
      puts("");

      /*
      **  Sort the array and display it again
      */

      qsort(ary, 5, sizeof(char *), compare);

      for (i = 0; i < 5; ++i)
            printf("%d - %s\n", i, ary[i]);

      return EXIT_SUCCESS;
}

int compare(const void *elem1, const void *elem2)
{
      char *e1 = *((char **)elem1), *e2 = *((char **)elem2);
      long l1, l2;

      l1 = (unsigned)strtol(&e1[posn], NULL, 16);
      l2 = (unsigned)strtol(&e2[posn], NULL, 16);
      return (l2 < l1) ? 1 : ((l1 == l2) ? 0 : -1);
}
```

TEXT STATISTICS

```
  2296 characters
    98 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     4 constants [character]
    15 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            22      47      55
    EXIT_SUCCESS            87
    NULL            44      52+     95      96
    UINT_MAX        67
    amalloc         52
    arg             27      32      33+     36      42
    argc            25      30
    argv            25      32
    ary             28      44      52      66      67      75      82      85
    compare         14      82      90
    ctype            9
    e1              92      95
    e2              92      96
    elem1           14      90      92
    elem2           14      90      92
    exit            22
    h        7       8       9      10
    i       27      64+     66+     67      74+     75+     84+     85+
    l1              93      95      97+
    l2              93      96      97+
    limits          10
    main            25
    mdalloc         44
    posn            15      66      67      95      96
    printf          35      75      85
    puts            19      20      21      46      54      76
    qsort           82
    random          67
    randomize       63
    sprintf         66      67
    stdio            7
    stdlib           8
    strtol          95      96
    toupper         32
    usage           17      31      34
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  This is a copyrighted work which is functionally identical to work
**  originally published in Micro Cornucopia magazine (issue #52, March-April,
**  1990) and is freely licensed by the author, Walter Bright, for any use.
*/

/*_ mem.c   Fri Jan 26 1990   Modified by: Walter Bright */
/* $Header: /proj/products/merlin/port/RCS/mem.c,v 1.19 89/10/20 14:36:02 bright Exp Locker: bright $
 */
/* Memory management package                    */

#if defined(VAX11C)
#define  __FILE__  "mem.c"
#endif

#include    <stdio.h>
#include    <stdlib.h>
#include    <io.h>

#ifndef MEM_H
#include    "mem.h"
#endif

#ifndef assert
#include    <assert.h>
#endif

#if defined(_MSC_VER)
#include    <dos.h>
#endif

#if !defined(VAX11C)
#ifdef BSDUNIX
#include <strings.h>
#else
#include <string.h>
#endif
#else
extern char *strcpy(),*memcpy();
extern int strlen();
#endif  /* VAX11C */

int mem_inited = 0;           /* != 0 if initialized              */

static int mem_behavior = MEM_ABORTMSG;
static int (*fp)() = (int (*)())NULL;    /* out-of-memory handler      */
static int mem_count;         /* # of allocs that haven't been free'd   */
static int mem_scount;        /* # of sallocs that haven't been free'd */
static int near mem_exception(); /* called when out of memory */

/* Determine where to send error messages */
#if defined(MSDOS) || defined(__MSDOS__)
#define ferr      stdout      /* stderr can't be redirected with MS-DOS */
#else
#define ferr      stderr
#endif

/******************************/

void mem_setexception(flag,handler_fp)
#if defined(__cplusplus) && __cplusplus
enum MEM_E flag;
#else
int flag;
#endif
int (*handler_fp)();
{
    mem_behavior = flag;
    fp = (mem_behavior == MEM_CALLFP) ? handler_fp : 0;
#if defined(MEM_DEBUG) && MEM_DEBUG
    assert(0 <= flag && flag <= MEM_RETRY);
#endif
}

/************************
 * This is called when we're out of memory.
 * Returns:
 *    1:    try again to allocate the memory
 *    0:    give up and return NULL
 */

static int near mem_exception()
{   int behavior;

    behavior = mem_behavior;
    while (1)
    {
      switch (behavior)
      {
          case MEM_ABORTMSG:
#if defined(MSDOS) || defined(__MSDOS__) || defined(__OS2__)
            /* Avoid linking in buffered I/O */
          { static char msg[] = "Fatal error: out of memory\r\n";

            write(1,msg,sizeof(msg) - 1);
          }
#else
            fputs("Fatal error: out of memory\n",ferr);
```

```
#endif
            /* FALL-THROUGH */
          case MEM_ABORT:
            exit(EXIT_FAILURE);
            /* NOTREACHED */
          case MEM_CALLFP:
            assert(fp);
            behavior = (*fp)();
            break;
          case MEM_RETNULL:
            return 0;
          case MEM_RETRY:
            return 1;
          default:
            assert(0);
      }
    }
}

/***************************/

#if defined(MEM_DEBUG) && MEM_DEBUG

#undef mem_strdup

char *mem_strdup(s)
const char *s;
{
      return mem_strdup_debug(s,__FILE__,__LINE__);
}

char *mem_strdup_debug(s,file,line)
char *file;
const char *s;
int line;
{
      char *p;

      p = s
          ? (char *) mem_malloc_debug((unsigned) strlen(s) + 1,file,line)
          : NULL;
      return p ? strcpy(p,s) : p;
}
#else
char *mem_strdup(s)
const char *s;
{
      char *p;

      p = s ? (char *) mem_malloc((unsigned) strlen(s) + 1) : NULL;
      return p ? strcpy(p,s) : p;
}

#endif /* MEM_DEBUG */

#ifdef MEM_DEBUG

static long mem_maxalloc;     /* max # of bytes allocated         */
static long mem_numalloc;     /* current # of bytes allocated           */

#define BEFOREVAL 0x12345678  /* value to detect underrun   */
#define AFTERVAL  0x87654321  /* value to detect overrun    */

#if SUN || SUN386
static long afterval = AFTERVAL;    /* so we can do &afterval     */
#endif

/* The following should be selected to give maximum probability that    */
/* pointers loaded with these values will cause an obvious crash. On    */
/* Unix machines, a large value will cause a segment fault.       */
/* MALLOCVAL is the value to set malloc'd data to.                */

#if MSDOS || __MSDOS__ || __OS2__
#define BADVAL          0xFF
#define MALLOCVAL 0xEE
#else
#define BADVAL          0x7A
#define MALLOCVAL 0xEE
#endif

/* Disable mapping macros     */
#undef      mem_malloc
#undef      mem_calloc
#undef      mem_realloc
#undef      mem_free

/* Create a list of all alloc'ed pointers, retaining info about where   */
/* each alloc came from. This is a real memory and speed hog, but who   */
/* cares when you've got obscure pointer bugs.                    */

static struct mem_debug
{     struct mh
      { struct mem_debug *Mnext;    /* next in list              */
        struct mem_debug *Mprev;    /* previous value in list     */
        char *Mfile;          /* filename of where allocated           */
        int Mline;            /* line number of where allocated   */
        unsigned Mnbytes;     /* size of the allocation           */
        long Mbeforeval;      /* detect underrun of data          */
      } m;
      char data[1];           /* the data actually allocated           */
```

```
} mem_alloclist =
{
   {  (struct mem_debug *) NULL,
      (struct mem_debug *) NULL,
      "noname",
      11111,
      0,
      BEFOREVAL
   },
   AFTERVAL
};

/* Convert from a void *to a mem_debug struct.  */
#define mem_ptrtodl(p)  ((struct mem_debug *) ((char *)p - sizeof(struct mh)))

/* Convert from a mem_debug struct to a mem_ptr.      */
#define mem_dltoptr(dl) ((void *) &((dl)->data[0]))

#define next            m.Mnext
#define prev            m.Mprev
#define file            m.Mfile
#define line            m.Mline
#define nbytes          m.Mnbytes
#define beforeval m.Mbeforeval

/*****************************
 * Set new value of file,line
 */

void mem_setnewfileline(ptr,fil,lin)
void *ptr;
char *fil;
int lin;
{
    struct mem_debug *dl;

    dl = mem_ptrtodl(ptr);
    dl->file = fil;
    dl->line = lin;
}

/*****************************
 * Print out struct mem_debug.
 */

static void near mem_printdl(dl)
struct mem_debug *dl;
{
#if LPTR
      fprintf(ferr,"alloc'd from file '%s' line %d nbytes %d ptr x%lx\n",
            dl->file,dl->line,dl->nbytes,mem_dltoptr(dl));
#else
      fprintf(ferr,"alloc'd from file '%s' line %d nbytes %d ptr x%x\n",
            dl->file,dl->line,dl->nbytes,mem_dltoptr(dl));
#endif
}

/*****************************
 * Print out file and line number.
 */

static void near mem_fillin(fil,lin)
char *fil;
int lin;
{
      fprintf(ferr,"File '%s' line %d\n",fil,lin);
      fflush(ferr);
}

/*****************************
 * If MEM_DEBUG is not on for some modules, these routines will get
 * called.
 */

void *mem_calloc(u)
unsigned u;
{
      return mem_calloc_debug(u,__FILE__,__LINE__);
}

void *mem_malloc(u)
unsigned u;
{
      return mem_malloc_debug(u,__FILE__,__LINE__);
}

void *mem_realloc(p,u)
void *p;
unsigned u;
{
      return mem_realloc_debug(p,u,__FILE__,__LINE__);
}

void mem_free(p)
void *p;
{
      mem_free_debug(p,__FILE__,__LINE__);
}

```

```
300 | /*************************/
301 |
302 | void mem_freefp(p)
303 | void *p;
304 | {
305 |       mem_free(p);
306 | }
307 |
308 | /***********************
309 |  * Debug versions of mem_calloc(), mem_free() and mem_realloc().
310 |  */
311 |
312 | void *mem_malloc_debug(n,fil,lin)
313 | unsigned n;
314 | char *fil;
315 | int lin;
316 | {   void *p;
317 |
318 |     p = mem_calloc_debug(n,fil,lin);
319 |     if (p)
320 |       memset(p,MALLOCVAL,n);
321 |     return p;
322 | }
323 |
324 | void *mem_calloc_debug(n,fil,lin)
325 | unsigned n;
326 | char *fil;
327 | int lin;
328 | {
329 |     struct mem_debug *dl;
330 |
331 |     do
332 |       dl = (struct mem_debug *)
333 |            calloc(sizeof(*dl) + n + sizeof(AFTERVAL) - 1,1);
334 |     while (dl == NULL && mem_exception());
335 |     if (dl == NULL)
336 |     {
337 | #if 0
338 |       printf("Insufficient memory for alloc of %d at ",n);
339 |       mem_fillin(fil,lin);
340 |       printf("Max allocated was: %ld\n",mem_maxalloc);
341 | #endif
342 |       return NULL;
343 |     }
344 |     dl->file = fil;
345 |     dl->line = lin;
346 |     dl->nbytes = n;
347 |     dl->beforeval = BEFOREVAL;
348 | #if SUN || SUN386 /* bus error if we store a long at an odd address */
349 |     memcpy(&(dl->data[n]),&afterval,sizeof(AFTERVAL));
350 | #else
351 |     *(long *) &(dl->data[n]) = AFTERVAL;
352 | #endif
353 |
354 |     /* Add dl to start of allocation list */
355 |     dl->next = mem_alloclist.next;
356 |     dl->prev = &mem_alloclist;
357 |     mem_alloclist.next = dl;
358 |     if (dl->next != NULL)
359 |       dl->next->prev = dl;
360 |
361 |     mem_count++;
362 |     mem_numalloc += n;
363 |     if (mem_numalloc > mem_maxalloc)
364 |       mem_maxalloc = mem_numalloc;
365 |     return mem_dltoptr(dl);
366 | }
367 |
368 | void mem_free_debug(ptr,fil,lin)
369 | void *ptr;
370 | char *fil;
371 | int lin;
372 | {
373 |       struct mem_debug *dl;
374 |
375 |       if (ptr == NULL)
376 |             return;
377 | #if 0
378 |       {     fprintf(ferr,"Freeing NULL pointer at ");
379 |             goto err;
380 |       }
381 | #endif
382 |       if (mem_count <= 0)
383 |       {     fprintf(ferr,"More frees than allocs at ");
384 |             goto err;
385 |       }
386 |       dl = mem_ptrtodl(ptr);
387 |       if (dl->beforeval != BEFOREVAL)
388 |       {
389 | #if LPTR
390 |             fprintf(ferr,"Pointer x%lx underrun\n",ptr);
391 | #else
392 |             fprintf(ferr,"Pointer x%x underrun\n",ptr);
393 | #endif
394 |             goto err2;
395 |       }
396 | #if SUN || SUN386 /* Bus error if we read a long from an odd address    */
397 |       if (memcmp(&dl->data[dl->nbytes],&afterval,sizeof(AFTERVAL)) != 0)
398 | #else
399 |       if (*(long *) &dl->data[dl->nbytes] != AFTERVAL)
```

```
#endif
      {
#if LPTR
            fprintf(ferr,"Pointer x%lx overrun\n",ptr);
#else
            fprintf(ferr,"Pointer x%x overrun\n",ptr);
#endif
            goto err2;
      }
      mem_numalloc -= dl->nbytes;
      if (mem_numalloc < 0)
      {     fprintf(ferr,"error: mem_numalloc = %ld, dl->nbytes = %d\n",
                  mem_numalloc,dl->nbytes);
            goto err2;
      }

      /* Remove dl from linked list */
      if (dl->prev)
            dl->prev->next = dl->next;
      if (dl->next)
            dl->next->prev = dl->prev;

      /* Stomp on the freed storage to help detect references     */
      /* after the storage was freed.                            */
      memset((void *) dl,BADVAL,sizeof(*dl) + dl->nbytes);
      mem_count--;

      /* Some compilers can detect errors in the heap.      */
#if defined(DLC)
      {     int i;
            i = free(dl);
            assert(i == 0);
      }
#else
      free((void *) dl);
#endif
      return;

err2:
      mem_printdl(dl);
err:
      fprintf(ferr,"free'd from ");
      mem_fillin(fil,lin);
      assert(0);
      /* NOTREACHED */
}

/******************
 * Debug version of mem_realloc().
 */

void *mem_realloc_debug(oldp,n,fil,lin)
void *oldp;
unsigned n;
char *fil;
int lin;
{   void *p;
    struct mem_debug *dl;

    if (n == 0)
    { mem_free_debug(oldp,fil,lin);
      p = NULL;
    }
    else if (oldp == NULL)
      p = mem_malloc_debug(n,fil,lin);
    else
    {
      p = mem_malloc_debug(n,fil,lin);
      if (p != NULL)
      {
          dl = mem_ptrtodl(oldp);
          if (dl->nbytes < n)
            n = dl->nbytes;
          memcpy(p,oldp,n);
          mem_free_debug(oldp,fil,lin);
      }
    }
    return p;
}

/**************************/

void mem_check()
{   register struct mem_debug *dl;

    for (dl = mem_alloclist.next; dl != NULL; dl = dl->next)
      mem_checkptr(mem_dltoptr(dl));
}

/**************************/

void mem_checkptr(p)
register void *p;
{   register struct mem_debug *dl;

    for (dl = mem_alloclist.next; dl != NULL; dl = dl->next)
    {
      if (p >= (void *) &(dl->data[0]) &&
          p < (void *)((char *)dl + sizeof(struct mem_debug)-1 + dl->nbytes))
          goto L1;
```

```
500 |     }
501 |     assert(0);
502 |
503 | L1:
504 |     dl = mem_ptrtodl(p);
505 |     if (dl->beforeval != BEFOREVAL)
506 |     {
507 | #if LPTR
508 |           fprintf(ferr,"Pointer x%lx underrun\n",p);
509 | #else
510 |           fprintf(ferr,"Pointer x%x underrun\n",p);
511 | #endif
512 |           goto err2;
513 |     }
514 | #if SUN || SUN386 /* Bus error if we read a long from an odd address    */
515 |     if (memcmp(&dl->data[dl->nbytes],&afterval,sizeof(AFTERVAL)) != 0)
516 | #else
517 |     if (*(long *) &dl->data[dl->nbytes] != AFTERVAL)
518 | #endif
519 |     {
520 | #if LPTR
521 |           fprintf(ferr,"Pointer x%lx overrun\n",p);
522 | #else
523 |           fprintf(ferr,"Pointer x%x overrun\n",p);
524 | #endif
525 |           goto err2;
526 |     }
527 |     return;
528 |
529 | err2:
530 |     mem_printdl(dl);
531 |     assert(0);
532 | }
533 |
534 | #else
535 |
536 | /*************************/
537 |
538 | void *mem_malloc(numbytes)
539 | unsigned numbytes;
540 | {     void *p;
541 |
542 |       if (numbytes == 0)
543 |             return NULL;
544 |       while (1)
545 |       {
546 |             p = malloc(numbytes);
547 |             if (p == NULL)
548 |             {     if (mem_exception())
549 |                         continue;
550 |             }
551 |             else
552 |                   mem_count++;
553 |             break;
554 |       }
555 |       /*printf("malloc(%d) = x%lx\n",numbytes,p);*/
556 |       return p;
557 | }
558 |
559 | /*************************/
560 |
561 | void *mem_calloc(numbytes)
562 | unsigned numbytes;
563 | {     void *p;
564 |
565 |       if (numbytes == 0)
566 |             return NULL;
567 |       while (1)
568 |       {
569 |             p = calloc(numbytes,1);
570 |             if (p == NULL)
571 |             {     if (mem_exception())
572 |                         continue;
573 |             }
574 |             else
575 |                   mem_count++;
576 |             break;
577 |       }
578 |       /*printf("calloc(%d) = x%lx\n",numbytes,p);*/
579 |       return p;
580 | }
581 |
582 | /*************************/
583 |
584 | void *mem_realloc(oldmem_ptr,newnumbytes)
585 | void *oldmem_ptr;
586 | unsigned newnumbytes;
587 | {   void *p;
588 |
589 |     if (oldmem_ptr == NULL)
590 |       p = mem_malloc(newnumbytes);
591 |     else if (newnumbytes == 0)
592 |     { mem_free(oldmem_ptr);
593 |       p = NULL;
594 |     }
595 |     else
596 |     {
597 |       do
598 |           p = realloc(oldmem_ptr,newnumbytes);
599 |       while (p == NULL && mem_exception());
```

```
600 |     }
601 |     /*printf("realloc(x%lx,%d) = x%lx\n",oldmem_ptr,newnumbytes,p);*/
602 |     return p;
603 | }
604 |
605 | /*************************/
606 |
607 | void mem_free(ptr)
608 | void *ptr;
609 | {
610 |     /*printf("free(x%lx)\n",ptr);*/
611 |     if (ptr != NULL)
612 |     { assert(mem_count > 0);
613 |       mem_count--;
614 | #if defined(DLC) && DLC
615 |       {     int i;
616 |
617 |             i = free(ptr);
618 |             assert(i == 0);
619 |       }
620 | #else
621 |       free(ptr);
622 | #endif
623 |     }
624 | }
625 |
626 | #endif /* MEM_DEBUG */
627 |
628 | /*************************/
629 |
630 | void mem_init()
631 | {
632 |       if (mem_inited == 0)
633 |       {     mem_count = 0;
634 | #if defined(MEM_DEBUG) && MEM_DEBUG
635 |             mem_numalloc = 0;
636 |             mem_maxalloc = 0;
637 |             mem_alloclist.next = NULL;
638 | #endif
639 | #if defined(__ZTC__) || defined(__SC__)
640 |             /* Necessary if mem_sfree() calls free() before any   */
641 |             /* calls to malloc().                                 */
642 |             free(malloc(1));  /* initialize storage allocator     */
643 | #endif
644 |             mem_inited++;
645 |       }
646 | }
647 |
648 | /*************************/
649 |
650 | void mem_term()
651 | {
652 |
653 |       if (mem_inited)
654 |       {
655 | #if defined(MEM_DEBUG) && MEM_DEBUG
656 |             register struct mem_debug *dl;
657 |
658 |             for (dl = mem_alloclist.next; dl; dl = dl->next)
659 |             {     fprintf(ferr,"Unfreed pointer: ");
660 |                   mem_printdl(dl);
661 |             }
662 | #if 0
663 |             fprintf(ferr,"Max amount ever allocated == %ld bytes\n",
664 |                   mem_maxalloc);
665 | #endif
666 | #else
667 |             if (mem_count)
668 |                   fprintf(ferr,"%d unfreed items\n",mem_count);
669 |             if (mem_scount)
670 |                   fprintf(ferr,"%d unfreed s items\n",mem_scount);
671 | #endif /* MEM_DEBUG */
672 |             assert(mem_count == 0 && mem_scount == 0);
673 |             mem_inited = 0;
674 |       }
675 | }
676 |
677 | #undef next
678 | #undef prev
679 | #undef file
680 | #undef line
681 | #undef nbytes
682 | #undef beforeval
```

TEXT STATISTICS

15982 characters
682 lines

LEXICAL STATISTICS

75 comments [std-C]
0 comments [C++]
115 preprocessor instructions
0 constants [character]
26 constants [string]
53 constants [numeric]

WARNINGS

line 153 : control character ignored

SYMBOL TABLE

AFTERVAL 161 164 209 333 349 351 397 399 515 517
BADVAL 173 176 424
BEFOREVAL 160 207 347 387 505
BSDUNIX 34
DLC 428 614+
EXIT_FAILURE 103
L1 499 503
LPTR 248 389 402 507 520
MALLOCVAL 174 177 320
MEM_ABORT 102
MEM_ABORTMSG 46 91
MEM_CALLFP 70 105
MEM_DEBUG 71+ 121+ 155 634+ 655+
MEM_E 63
MEM_H 21
MEM_RETNULL 109
MEM_RETRY 72 111
MSDOS 53 92 172
Mbeforeval 197 223
Mfile 194 220
Mline 195 221
Mnbytes 196 222
Mnext 192 218
Mprev 193 219
NULL 47 140 149 202 203 334 335 342 358 375 461
463 468 485 495 543 547 566 570 589 593 599 611
637
SUN 163 348 396 514
SUN386 163 348 396 514
VAX11C 13 33
_MSC_VER 29
__FILE__ 14 128 277 283 290 296
__LINE__ 128 277 283 290 296
__MSDOS__ 53 92 172
__OS2__ 92 172
__SC__ 639
__ZTC__ 639
__cplusplus 62+
afterval 164 349 397 515
assert 25 26 72 106 114 431 443 501 531 612 618
672
beforeval 223 347 387 505 682
behavior 84 86 89 107
calloc 333 569
data 199 216 349 351 397 399 497 515 517
defined 13 29 33 53+ 62 71 92+ 121 428 614 634
639+ 655
dl 216+ 234 236 237 238 245 246 250+ 253+ 329 332
333 334 335 344 345 346 347 349 351 355 356 357
358 359+ 365 373 386 387 397+ 399+ 409 412 417 418+
419 420+ 424+ 430 434 439 457 470 471 472 483 485+
486 493 495+ 497 498+ 504 505 515+ 517+ 530 656 658+
660
dos 30
err 379 384 440
err2 394 407 413 438 512 525 529
exit 103
ferr 54 56 99 249 252 265 266 378 383 390 392
403 405 411 441 508 510 521 523 659 663 668 670
fflush 266
fil 229 231 237 261 262 265 312 314 318 324 326
339 344 368 370 442 451 454 460 464 467 474
file 131 132 139 220 237 250 253 344 679
flag 61 63 65 69 72+
fp 47 70 106 107
fprintf 249 252 265 378 383 390 392 403 405 411 441
508 510 521 523 659 663 668 670
fputs 99
free 430 434 617 621 642
h 17 18 19 26 30 35 37
handler_fp 61 67 70
i 429 430 431 615 617 618
io 19
lin 229 232 238 261 263 265 312 315 318 324 327
339 345 368 371 442 451 455 460 464 467 474
line 131 134 139 221 238 250 253 345 680
m 198 218 219 220 221 222 223
malloc 546 642
mem_alloclist 200 355 356 357 485 495 637 658
mem_behavior 46 69 70 86

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  This is a copyrighted work which is functionally identical to work
**  originally published in Micro Cornucopia magazine (issue #52, March-April,
**  1990) and is freely licensed by the author, Walter Bright, for any use.
*/

/*_ mem.h   Fri May 26 1989   Modified by: Walter Bright */
/* $Header: /proj/products/merlin/port/RCS/mem.h,v 1.11 89/10/23 11:39:00 bright Exp $ */
/* Copyright 1986-1988 by Northwest Software    */
/* All Rights Reserved                  */
/* Written by Walter Bright             */

#ifndef MEM_H
#define MEM_H     1

#ifndef TOOLKIT_H
#include    "toolkit.h"
#endif

/*
 * Memory management routines.
 *
 * Compiling:
 *
 *    #define MEM_DEBUG 1 when compiling to enable extended debugging
 *    features.
 *
 * Features always enabled:
 *
 *    o mem_init() is called at startup, and mem_term() at
 *      close, which checks to see that the number of alloc's is
 *      the same as the number of free's.
 *    o Behavior on out-of-memory conditions can be controlled
 *      via mem_setexception().
 *
 * Extended debugging features:
 *
 *    o Enabled by #define MEM_DEBUG 1 when compiling.
 *    o Check values are inserted before and after the alloc'ed data
 *      to detect pointer underruns and overruns.
 *    o Free'd pointers are checked against alloc'ed pointers.
 *    o Free'd storage is cleared to smoke out references to free'd data.
 *    o Realloc'd pointers are always changed, and the previous storage
 *      is cleared, to detect erroneous dependencies on the previous
 *      pointer.
 *    o The routine mem_checkptr() is provided to check an alloc'ed
 *      pointer.
 */

/******************** GLOBAL VARIABLES ************************/

extern int mem_inited;        /* != 0 if mem package is initialized.    */
                        /* Test this if you have other packages   */
                        /* that depend on mem being initialized   */

/******************** PUBLIC FUNCTIONS ************************/

/*********************************
 * Set behavior when mem runs out of memory.
 * Input:
 *    flag =      MEM_ABORTMSG:     Abort the program with the message
 *                      'Fatal error: out of memory' sent
 *                      to stdout. This is the default behavior.
 *          MEM_ABORT:  Abort the program with no message.
 *          MEM_RETNULL:      Return NULL back to caller.
 *          MEM_CALLFP: Call application-specified function.
 *                      fp must be supplied.
 *    fp                Optional function pointer. Supplied if
 *                      (flag == MEM_CALLFP). This function returns
 *                      MEM_XXXXX, indicating what mem should do next.
 *                      The function could do things like swap
 *                      data out to disk to free up more memory.
 *    fp could also return:
 *          MEM_RETRY:  Try again to allocate the space. Be
 *                      careful not to go into an infinite loop.
 */

#if defined(__cplusplus) && __cplusplus
enum MEM_E { MEM_ABORTMSG, MEM_ABORT, MEM_RETNULL, MEM_CALLFP, MEM_RETRY };
void mem_setexception P((enum MEM_E, int (*)()));
#else
#define MEM_ABORTMSG    0
#define MEM_ABORT 1
#define MEM_RETNULL     2
#define MEM_CALLFP      3
#define MEM_RETRY 4
void mem_setexception P((int, int(*)()));
#endif


/***************************
 * Allocate space for string, copy string into it, and
 * return pointer to the new string.
 * This routine doesn't really belong here, but it is used so often
 * that I gave up and put it here.
 * Use:
 *    char *mem_strdup(const char *s);
 * Returns:
```

```
 *    pointer to copied string if successful.
 *    else returns NULL (if MEM_RETNULL)
 */

char *mem_strdup P((const char *));

/*************************
 * Function so we can have a pointer to function mem_free().
 * This is needed since mem_free is sometimes defined as a macro,
 * and then the preprocessor screws up.
 * The pointer to mem_free() is used frequently with the list package.
 * Use:
 *    void mem_freefp(void *p);
 */

/*************************
 * Check for errors. This routine does a consistency check on the
 * storage allocator, looking for corrupted data. It should be called
 * when the application has CPU cycles to burn.
 * Use:
 *    void mem_check(void);
 */

void mem_check P((void ));

/*************************
 * Check ptr to see if it is in the range of allocated data.
 * Cause assertion failure if it isn't.
 */

void mem_checkptr P((void *ptr));

/*************************
 * Allocate and return a pointer to numbytes of storage.
 * Use:
 *    void *mem_malloc(unsigned numbytes);
 *    void *mem_calloc(unsigned numbytes); allocated memory is cleared
 * Input:
 *    numbytes    Number of bytes to allocate
 * Returns:
 *    if (numbytes > 0)
 *          pointer to allocated data, NULL if out of memory
 *    else
 *          return NULL
 */

void *mem_malloc P((unsigned));
void *mem_calloc P((unsigned));

/*****************************
 * Reallocate memory.
 * Use:
 *    void *mem_realloc(void *ptr,unsigned numbytes);
 */

void *mem_realloc P((void *,unsigned));

/*****************************
 * Free memory allocated by mem_malloc(), mem_calloc() or mem_realloc().
 * Use:
 *    void mem_free(void *ptr);
 */

void mem_free P((void *));

/*************************
 * Initialize memory handler.
 * Use:
 *    void mem_init(void);
 * Output:
 *    mem_inited = 1
 */

void mem_init P((void ));

/*************************
 * Terminate memory handler. Useful for checking for errors.
 * Use:
 *    void mem_term(void);
 * Output:
 *    mem_inited = 0
 */

void mem_term P((void ));

/* The following stuff forms the implementation rather than the
 * definition, so ignore it.
 */

#if defined(MEM_DEBUG) && MEM_DEBUG       /* if creating debug version  */
#define mem_strdup(p)   mem_strdup_debug((p),__FILE__,__LINE__)
#define mem_malloc(u)   mem_malloc_debug((u),__FILE__,__LINE__)
#define mem_calloc(u)   mem_calloc_debug((u),__FILE__,__LINE__)
#define mem_realloc(p,u)      mem_realloc_debug((p),(u),__FILE__,__LINE__)
#define mem_free(p)     mem_free_debug((p),__FILE__,__LINE__)

char *mem_strdup_debug  P((const char *,char *,int));
void *mem_calloc_debug  P((unsigned,char *,int));
void *mem_malloc_debug  P((unsigned,char *,int));
void *mem_realloc_debug P((void *,unsigned,char *,int));
```

```
 201 | void  mem_free_debug    P((void *,char *,int));
 202 | void  mem_freefp  P((void *));
 203 |
 204 | void mem_setnewfileline P((void *,char *,int));
 205 |
 206 | #else
 207 |
 208 | #define mem_freefp      mem_free
 209 | #define mem_check()
 210 | #define mem_checkptr(p)
 211 |
 212 | #endif /* MEM_DEBUG */
 213 |
 214 | #endif /* MEM_H */


TEXT STATISTICS

  6885 characters
   214 lines

LEXICAL STATISTICS

    27 comments [std-C]
     0 comments [C++]
    25 preprocessor instructions
     0 constants [character]
     1 constants [string]
     6 constants [numeric]

WARNINGS

     line    56 : control character ignored

SYMBOL TABLE

    MEM_ABORT       81      85
    MEM_ABORTMSG            81      84
    MEM_CALLFP              81      87
    MEM_DEBUG      190+
    MEM_E           81      82
    MEM_H           15      16
    MEM_RETNULL             81      86
    MEM_RETRY       81      88
    P       82      89     105     124     131     147     148     156     164     174     184     197
           198     199     200     201     202     204
    TOOLKIT_H       18
    __FILE__       191     192     193     194     195
    __LINE__       191     192     193     194     195
    __cplusplus             80+
    defined         80     190
    mem_calloc             148     193
    mem_calloc_debug       193     198
    mem_check      124     209
    mem_checkptr           131     210
    mem_free       164     195     208
    mem_free_debug         195     201
    mem_freefp             202     208
    mem_init       174
    mem_inited              54
    mem_malloc             147     192
    mem_malloc_debug       192     199
    mem_realloc            156     194
    mem_realloc_debug      194     200
    mem_setexception        82      89
    mem_setnewfileline             204
    mem_strdup             105     191
    mem_strdup_debug       191     197
    mem_term       184
    p      191+    194+    195+    210
    ptr            131
    u      192+    193+    194+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  MEMAVAIL.C - Report available DOS memory
  5 | **
  6 | **  public domain by Thor Johnson
  7 | */
  8 |
  9 | #include <dos.h>
 10 | #include "snpdosys.h"
 11 |
 12 | long memavail(void)
 13 | {
 14 |       union REGS regs;
 15 |
 16 |       /* Request impossibly large number of 16-byte paragraphs from DOS */
 17 |
 18 |       regs.h.ah = 0x48;
 19 |       regs.x.bx = 0xFFFF;
 20 |
 21 |       int86(0x21,&regs,&regs);
 22 |
 23 |       return((long)regs.x.bx * 16L);
 24 | }
 25 |
 26 | #ifdef TEST
 27 |
 28 | #include <stdio.h>
 29 |
 30 | main()
 31 | {
 32 |       printf("Available DOS memory = %ld bytes\n", memavail());
 33 |       return 0;
 34 | }
 35 |
 36 | #endif /* TEST */
```

TEXT STATISTICS

```
   595 characters
    36 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    REGS            14
    TEST            26
    ah              18
    bx              19      23
    dos              9
    h        9      18      28
    int86           21
    main            30
    memavail        12      32
    printf          32
    regs            14      18      19      21+     23
    stdio           28
    x       19      23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MEMMEM.C - A strstr() work-alike for non-text buffers
**
**  public domain by Bob Stout
**
**  Warning: The memchr() in Borland C/C++ versions *prior* to 4.x is broken!
*/

#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

void *memmem(const void *buf, const void *pattern, size_t buflen, size_t len)
{
#if defined(__TURBOC__) && (__TURBOC__ >= 0x500)

      char *bf = (char *)buf, *pt = (char *)pattern, *p = bf;

      while (len <= (buflen - (p - bf)))
      {
            if (NULL != (p = memchr(p, (int)(*pt), buflen - (p - bf))))
            {
                  if (Success_ == memcmp(p, pattern, len))
                        return p;
                  else  ++p;
            }
            else  break;
      }
      return NULL;

#else /* Borland/Turbo C/C++ version prior to 4.x */

      size_t i, j;
      char *bf = (char *)buf, *pt = (char *)pattern;

      if (len > buflen)
            return (void *)NULL;

      for (i = 0; i <= (buflen - len); ++i)
      {
            for (j = 0; j < len; ++j)
            {
                  if (pt[j] != bf[i + j])
                        break;
            }
            if (j == len)
                  return (bf + i);
      }
      return NULL;

#endif
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

main()
{
      char buf[13] = "\0""12344567890\x1b";
      char a[3] = "456";
      char b[3] = "\0""12";
      char c[3] = "90\x1b";
      char d[3] = "ABC";
      char e[3] = "0\x1b""\0";
      char f[1] = "\x1b";
      char *ptr;

      if (NULL == (ptr = memmem(buf, a, 13, 3)))
            puts("a not found in buf");
      else  printf("a found in buf at posn %d\n", ptr - buf);

      if (NULL == (ptr = memmem(buf, b, 13, 3)))
            puts("b not found in buf");
      else  printf("b found in buf at posn %d\n", ptr - buf);

      if (NULL == (ptr = memmem(buf, c, 13, 3)))
            puts("c not found in buf");
      else  printf("c found in buf at posn %d\n", ptr - buf);

      if (NULL == (ptr = memmem(buf, d, 13, 3)))
            puts("d not found in buf");
      else  printf("d found in buf at posn %d\n", ptr - buf);

      if (NULL == (ptr = memmem(buf, e, 13, 3)))
            puts("e not found in buf");
      else  printf("e found in buf at posn %d\n", ptr - buf);

      if (NULL == (ptr = memmem(buf, f, 13, 1)))
            puts("f not found in buf");
      else  printf("f found in buf at posn %d\n", ptr - buf);
```

```
 101 |
 102 |       return 0;
 103 | }
 104 |
 105 | #endif /* TEST */
```

TEXT STATISTICS

```
  2697 characters
   105 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
    24 constants [string]
    23 constants [numeric]
```

SYMBOL TABLE

```
    NULL            26      34      42      54      78      82      86      90      94      98
    Success_        28
    TEST            63
    __TURBOC__              20+
    __cplusplus             14+     59+
    a       70      78
    b       71      82
    bf              22+     24      26      39      48      52
    buf             18      22      39      69      78      80      82      84      86      88      90
            92      94      96      98     100
    buflen          18      24      26      41      44
    c       72      86
    d       73      90
    defined         14      20      59
    e       74      94
    f       75      98
    h       11      65
    i       38      44+     48      52
    j       38      46+     48+     51
    len             18      24      28      41      44      46      51
    main            67
    memchr          26
    memcmp          28
    memmem          18      78      82      86      90      94      98
    p       22      24      26+     28      29      30
    pattern         18      22      28      39
    printf          80      84      88      92      96     100
    pt              22      26      39      48
    ptr             76      78      80      82      84      86      88      90      92      94      96
            98     100
    puts            79      83      87      91      95      99
    size_t          18+     38
    stdio           65
    string          11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Public domain demo by Ray Gardner, 7 dec 88
**
**  Here's an old programming trick that (I bet) will be new to at least a
**  few of you out there, even some "old hands".  I don't remember where I
**  saw this; it might have been Jon Bentley's "Programming Pearls" column in
**  Communications of the ACM.
**
**  Have you ever wanted to exchange two adjacent areas of storage which
**  might be of two different lengths?  There are some tricky and complicated
**  "efficient" methods to do this without using a lot of extra temporary
**  storage.  But there is also an old and simple way: Assume that the buffer
**  looks like this:
**
**     |...... head .......|.................. tail .................|
**
**  You reverse the head, reverse the tail, then reverse the entire buffer.
**  That's all there is to it.  It will leave you with:
**
**     |.................. tail .................|...... head .......|
**
**  Here's code:
*/

#include <stdlib.h>
#include "memrev.h"

/*
**  reverse "count" bytes starting at "buf"
*/

void memrev(char  *buf, size_t count)
{
      char *r;

      for (r = buf + count - 1; buf < r; buf++, r--)
      {
            *buf ^= *r;
            *r   ^= *buf;
            *buf ^= *r;
      }
}

/*
**  swap "head" bytes with "tail" bytes at "buf"
*/

void aswap(char  *buf, size_t head, size_t tail)
{
      memrev(buf, head);
      memrev(buf + head, tail);
      memrev(buf, head + tail);
}
```

TEXT STATISTICS

```
  1536 characters
    55 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    aswap           50
    buf             34      38+     40      41      42      50      52      53      54
    count           34      38
    h       27
    head            50      52      53      54
    memrev          34      52      53      54
    r       36      38+     40      41      42
    size_t          34      50+
    stdlib          27
    tail            50      53      54
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header file for MEMREV.C
   5 | */
   6 |
   7 | #ifndef MEMREV__H
   8 | #define MEMREV__H
   9 |
  10 | #include <stddef.h>
  11 |
  12 | void memrev(char  *buf, size_t count);
  13 | void aswap(char  *buf, size_t head, size_t tail);
  14 |
  15 | #endif /* MEMREV__H */
```

TEXT STATISTICS

```
   273 characters
    15 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    MEMREV__H        7       8
    aswap           13
    buf             12      13
    count           12
    h       10
    head            13
    memrev          12
    size_t          12      13+
    stddef          10
    tail            13
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  METAPHON.C - Phonetic string matching
**
**  The Metaphone algorithm was developed by Lawrence Phillips. Like the
**  Soundex algorithm, it compares words that sound alike but are spelled
**  differently. Metaphone was designed to overcome difficulties encountered
**  with Soundex.
**
**  This implementation was written by Gary A. Parker and originally published
**  in the June/July, 1991 (vol. 5 nr. 4) issue of C Gazette. As published,
**  this code was explicitly placed in the public domain by the author.
*/

#include <ctype.h>
#include "phonetic.h"

/*
**  Character coding array
*/

static char vsvfn[26] = {
      1,16,4,16,9,2,4,16,9,2,0,2,2,2,1,4,0,2,4,4,1,0,0,0,8,0};
/*    A  B C  D E F G  H I J K L M N O P Q R S T U V W X Y Z      */

/*
**  Macros to access the character coding array
*/

#define vowel(x)  (vsvfn[(x) - 'A'] & 1)  /* AEIOU    */
#define same(x)   (vsvfn[(x) - 'A'] & 2)  /* FJLMNR   */
#define varson(x) (vsvfn[(x) - 'A'] & 4)  /* CGPST    */
#define frontv(x) (vsvfn[(x) - 'A'] & 8)  /* EIY      */
#define noghf(x)  (vsvfn[(x) - 'A'] & 16) /* BDH      */

/*
**  metaphone()
**
**  Arguments: 1 - The word to be converted to a metaphone code.
**             2 - A MAXMETAPH+1 char field for the result.
**             3 - Function flag:
**                 If 0: Compute the Metaphone code for the first argument,
**                       then compare it to the Metaphone code passed in
**                       the second argument.
**                 If 1: Compute the Metaphone code for the first argument,
**                       then store the result in the area pointed to by the
**                       second argument.
**
**  Returns: If function code is 0, returns Success_ for a match, else Error_.
**           If function code is 1, returns Success_.
*/

Boolean_T metaphone(const char *Word, char *Metaph, metaphlag Flag)
{
      char *n, *n_start, *n_end;    /* Pointers to string              */
      char *metaph, *metaph_end;    /* Pointers to metaph              */
      char ntrans[512];             /* Word with uppercase letters     */
      char newm[MAXMETAPH + 4];     /* New metaph for comparison       */
      int KSflag;                   /* State flag for X translation    */

      /*
      ** Copy word to internal buffer, dropping non-alphabetic characters
      ** and converting to upper case.
      */

      for (n = ntrans + 1, n_end = ntrans + sizeof(ntrans) - 2;
            *Word && n < n_end; ++Word)
      {
            if (isalpha(*Word))
                  *n++ = toupper(*Word);
      }

      if (n == ntrans + 1)
            return Error_;           /* Return if zero characters       */
      else  n_end = n;              /* Set end of string pointer       */

      /*
      ** Pad with NULs, front and rear
      */

      *n++ = NUL;
      *n   = NUL;
      n    = ntrans;
      *n++ = NUL;

      /*
      ** If doing comparison, redirect pointers
      */

      if (COMPARE == Flag)
      {
            metaph = Metaph;
            Metaph = newm;
      }

      /*
      ** Check for PN, KN, GN, WR, WH, and X at start
      */

```

```
101 |       switch (*n)
102 |       {
103 |       case 'P':
104 |       case 'K':
105 |       case 'G':
106 |             if ('N' == *(n + 1))
107 |                   *n++ = NUL;
108 |             break;
109 |
110 |       case 'A':
111 |             if ('E' == *(n + 1))
112 |                   *n++ = NUL;
113 |             break;
114 |
115 |       case 'W':
116 |             if ('R' == *(n + 1))
117 |                   *n++ = NUL;
118 |             else if ('H' == *(n + 1))
119 |             {
120 |                   *(n + 1) = *n;
121 |                   *n++ = NUL;
122 |             }
123 |             break;
124 |
125 |       case 'X':
126 |             *n = 'S';
127 |             break;
128 |       }
129 |
130 |       /*
131 |       ** Now loop through the string, stopping at the end of the string
132 |       ** or when the computed Metaphone code is MAXMETAPH characters long.
133 |       */
134 |
135 |       KSflag = False_;              /* State flag for KStranslation     */
136 |       for (metaph_end = Metaph + MAXMETAPH, n_start = n;
137 |             n <= n_end && Metaph < metaph_end; ++n)
138 |       {
139 |             if (KSflag)
140 |             {
141 |                   KSflag = False_;
142 |                   *Metaph++ = *n;
143 |             }
144 |             else
145 |             {
146 |                   /* Drop duplicates except for CC    */
147 |
148 |                   if (*(n - 1) == *n && *n != 'C')
149 |                         continue;
150 |
151 |                   /* Check for F J L M N R  or first letter vowel */
152 |
153 |                   if (same(*n) || (n == n_start && vowel(*n)))
154 |                         *Metaph++ = *n;
155 |                   else switch (*n)
156 |                   {
157 |                   case 'B':
158 |                         if (n < n_end || *(n - 1) != 'M')
159 |                               *Metaph++ = *n;
160 |                         break;
161 |
162 |                   case 'C':
163 |                         if (*(n - 1) != 'S' || !frontv(*(n + 1)))
164 |                         {
165 |                               if ('I' == *(n + 1) && 'A' == *(n + 2))
166 |                                     *Metaph++ = 'X';
167 |                               else if (frontv(*(n + 1)))
168 |                                     *Metaph++ = 'S';
169 |                               else if ('H' == *(n + 1))
170 |                                     *Metaph++ = ((n == n_start &&
171 |                                           !vowel(*(n + 2))) ||
172 |                                           'S' == *(n - 1)) ? 'K' : 'X';
173 |                               else  *Metaph++ = 'K';
174 |                         }
175 |                         break;
176 |
177 |                   case 'D':
178 |                         *Metaph++ = ('G' == *(n + 1) && frontv(*(n + 2))) ?
179 |                               'J' : 'T';
180 |                         break;
181 |
182 |                   case 'G':
183 |                         if ((*(n + 1) != 'H' || vowel(*(n + 2))) &&
184 |                               (*(n + 1) != 'N' || ((n + 1) < n_end &&
185 |                               (*(n + 2) != 'E' || *(n + 3) != 'D'))) &&
186 |                               (*(n - 1) != 'D' || !frontv(*(n + 1))))
187 |                         {
188 |                               *Metaph++ = (frontv(*(n + 1)) &&
189 |                                     *(n + 2) != 'G') ? 'J' : 'K';
190 |                         }
191 |                         else if ('H' == *(n + 1) && !noghf(*(n - 3)) &&
192 |                               *(n - 4) != 'H')
193 |                         {
194 |                               *Metaph++ = 'F';
195 |                         }
196 |                         break;
197 |
198 |                   case 'H':
199 |                         if (!varson(*(n - 1)) && (!vowel(*(n - 1)) ||
200 |                               vowel(*(n + 1))))
```

```
                        {
                              *Metaph++ = 'H';
                        }
                        break;

                  case 'K':
                        if (*(n - 1) != 'C')
                              *Metaph++ = 'K';
                        break;

                  case 'P':
                        *Metaph++ = ('H' == *(n + 1)) ? 'F' : 'P';
                        break;

                  case 'Q':
                        *Metaph++ = 'K';
                        break;

                  case 'S':
                        *Metaph++ = ('H' == *(n + 1) || ('I' == *(n + 1) &&
                              ('O' == *(n + 2) || 'A' == *(n + 2)))) ?
                              'X' : 'S';
                        break;

                  case 'T':
                        if ('I' == *(n + 1) && ('O' == *(n + 2) ||
                              'A' == *(n + 2)))
                        {
                              *Metaph++ = 'X';
                        }
                        else if ('H' == *(n + 1))
                              *Metaph++ = '0';
                        else if (*(n + 1) != 'C' || *(n + 2) != 'H')
                              *Metaph++ = 'T';
                        break;

                  case 'V':
                        *Metaph++ = 'F';
                        break;

                  case 'W':
                  case 'Y':
                        if (vowel(*(n + 1)))
                              *Metaph++ = *n;
                        break;

                  case 'X':
                        if (n == n_start)
                              *Metaph++ = 'S';
                        else
                        {
                              *Metaph++ = 'K';
                              KSflag = True_;
                        }
                        break;

                  case 'Z':
                        *Metaph++ = 'S';
                        break;
                  }
            }

            /*
            ** Compare new Metaphone code with old
            */

            if (COMPARE == Flag &&
                  *(Metaph - 1) != metaph[(Metaph - newm) - 1])
            {
                  return Error_;
            }
      }

      /*
      ** If comparing, check if Metaphone codes were equal in length
      */

      if (COMPARE == Flag && metaph[Metaph - newm])
            return Error_;

      *Metaph = NUL;
      return Success_;
}

#ifdef TEST

/*
**  Test demo to search a drive for a filename which sounds similar to
**  file names passed on the command line
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dirent.h"     /* metaphone() is portable so use Posix   */
#include "filnames.h"   /* valid_fname() prototype                */
#include "snip_str.h"   /* strchcat() prototype                   */

/*
**  Prototypes
```

```
301 | */
302 |
303 | void print_find_t(char * dir, DOSFileData *find);
304 | void search(char *dir, char *name);
305 |
306 | Boolean_T found = False_;
307 | char meta[MAXMETAPH + 4];
308 |
309 | main(int argc, char *argv[])
310 | {
311 |       char *ptr;
312 |
313 |       if (2 > argc)
314 |       {
315 |             puts("Usage: METAPHON filename [...filename]");
316 |             return EXIT_FAILURE;
317 |       }
318 |       while (--argc)
319 |       {
320 |             char *fname = *++argv;
321 |
322 |             if (Success_ != valid_fname(fname, -1))
323 |             {
324 |                   printf("\nIllegal filename: %s\n", fname);
325 |                   continue;
326 |             }
327 |             printf("\nSearching for: %s\n", fname);
328 |
329 |             /* Remove the extension, if any     */
330 |
331 |             if (NULL != (ptr = strchr(fname, '.')))
332 |                   *ptr = NUL;
333 |
334 |             /* Store the Metaphone code         */
335 |
336 |             metaphone(fname, meta, GENERATE);
337 |             printf("Metaphone for %s is %s\n", fname, meta);
338 |
339 |             /* Search for matches               */
340 |
341 |             search("", "");
342 |             if (!found)
343 |                   puts("A match was not found");
344 |       }
345 |       return EXIT_SUCCESS;
346 | }
347 |
348 | void search(char *dir, char *newdir)
349 | {
350 |       char         curdir[FILENAME_MAX];
351 |       DIR         *dirp;
352 |       DOSFileData *dstruct;
353 |       char        *ptr;
354 |       Boolean_T    retval;
355 |
356 |       while (NULL != (ptr = strchr(dir, '\\')))
357 |             *ptr = '/';
358 |
359 |       strcpy(curdir, dir);
360 |
361 |       if ('/' != LAST_CHAR(curdir))
362 |             strchcat(curdir, '/', FILENAME_MAX);
363 |
364 |       strcat(curdir, newdir);
365 |
366 |       if (NULL != (dirp = opendir(curdir)))
367 |       {
368 |             while (NULL != (dstruct = readdir(dirp)))
369 |             {
370 |                   if ('.' == *(ff_name(dstruct)))
371 |                         continue;
372 |                   else
373 |                   {
374 |                         /* Don't look at file extension     */
375 |
376 |                         if (NULL != (ptr = strchr(ff_name(dstruct), '.')))
377 |                               *ptr = NUL;
378 |                         else  ptr  = NULL;
379 |
380 |                         retval = metaphone(ff_name(dstruct), meta, COMPARE);
381 |
382 |                         /* Restore extension, if any        */
383 |
384 |                         if (ptr != NULL)
385 |                               *ptr = '.';
386 |
387 |                         if (Success_ == retval)
388 |                               print_find_t(curdir, dstruct);
389 |                   }
390 |                   if (ff_attr(dstruct) & _A_SUBDIR)
391 |                         search(curdir, ff_name(dstruct));
392 |             }
393 |             closedir(dirp);
394 |       }
395 | }
396 |
397 | void print_find_t(char *dir, DOSFileData *find)
398 | {
399 |       printf("%s/%-12s %8ld %2.2d-%02.2d-%02.2d %c%c%c%c%c\n", dir,
400 |             ff_name(find), ff_size(find), ff_mo(find),
```

```
 401 |             ff_day(find), (ff_yr(find) + 80) % 100,
 402 |             (ff_attr(find) & _A_SUBDIR) ? 'D' : '.',
 403 |             (ff_attr(find) & _A_RDONLY) ? 'R' : '.',
 404 |             (ff_attr(find) & _A_HIDDEN) ? 'H' : '.',
 405 |             (ff_attr(find) & _A_SYSTEM) ? 'S' : '.',
 406 |             (ff_attr(find) & _A_ARCH)   ? 'A' : '.');
 407 |       found = True_;
 408 | }
 409 |
 410 | #endif /* TEST */
```

TEXT STATISTICS

```
 12856 characters
   410 lines
```

LEXICAL STATISTICS

```
    39 comments [std-C]
     0 comments [C++]
    15 preprocessor instructions
   101 constants [character]
    12 constants [string]
    90 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       54     306     354
    COMPARE         91     267     278     380
    DIR            351
    DOSFileData             303     352     397
    EXIT_FAILURE            316
    EXIT_SUCCESS            345
    Error_          75     270     279
    FILENAME_MAX            350     362
    False_         135     141     306
    Flag            54      91     267     278
    GENERATE       336
    KSflag          60     135     139     141     253
    LAST_CHAR      361
    MAXMETAPH       59     136     307
    Metaph          54      93      94     136     137     142     154     159     166     168     170
           173     178     188     194     202     208     212     216     220     229     232     234
           238     244     249     252     258     268+    278     281
    NUL             82      83      85     107     112     117     121     281     332     377
    NULL           331     356     366     368     376     378     384
    Success_       282     322     387
    TEST           285
    True_          253     407
    Word            54      68+     70      71
    _A_ARCH        406
    _A_HIDDEN      404
    _A_RDONLY      403
    _A_SUBDIR      390     402
    _A_SYSTEM      405
    argc           309     313     318
    argv           309     320
    closedir       393
    ctype           16
    curdir         350     359     361     362     364     366     388     391
    dir            303     304     348     356     359     397     399
    dirp           351     366     368     393
    dstruct        352     368     370     376     380     388     390     391
    ff_attr        390     402     403     404     405     406
    ff_day         401
    ff_mo          400
    ff_name        370     376     380     391     400
    ff_size        400
    ff_yr          401
    find           303     397     400+    401+    402     403     404     405     406
    fname          320     322     324     327     331     336     337
    found          306     342     407
    frontv          34     163     167     178     186     188
    h       16     292     293     294
    isalpha         70
    main           309
    meta           307     336     337     380
    metaph          57      93     268     278
    metaph_end              57     136     137
    metaphlag       54
    metaphone       54     336     380
    n       56      67      68      71      74      76      82      83      84      85     101     106
           107     111     112     116     117     118     120+    121     126     136     137+    142
           148+    153+    154     155     158+    159     163+    165+    167     169     170     171
           172     178+    183+    184+    185+    186+    188     189     191+    192     199+    200
           207     212     220+    221+    226+    227     231     233+    243     244     248
    n_end           56      67      68      76     137     158     184
    n_start         56     136     153     170     248
    name           304
    newdir         348     364
    newm            59      94     268     278
    noghf           35     191
    ntrans          58      67+     74      84
    opendir        366
    print_find_t            303     388     397
    printf         324     327     337     399
    ptr            311     331     332     353     356     357     376     377     378     384     385
    puts           315     343
    readdir        368
```

```
retval          354     380     387
same             32     153
search          304     341     348     391
stdio           292
stdlib          293
strcat          364
strchcat        362
strchr          331     356     376
strcpy          359
string          294
toupper          71
valid_fname             322
varson           33     199
vowel            31     153     171     183     199     200     243
vsvfn            23      31      32      33      34      35
x       31+     32+     33+     34+     35+
```

```
/* +++Date last modified: 05-Jul-1997 */

/**************************************************
*
*         ANSI C Metric Conversion Macros
*              Mike Smith, Freeware
*   6 Grenville Road, Plymouth, England, PL4 9PX
*
*             mike@msmith.demon.co.uk
*
*   Version: 1.03
*   History: 5th of May, 1997
*
*   1.01 Taken out formulas and added values only Mon 17th April 1995
*   1.02 Error corrections Sat 13th July 1996
*   1.03 Added macros 5th of May, 1997
*
*================================================
*  While I have done my best to assure accuracy
*     I can not confirm these are accurate.
*
*  This was done as part of a program which I
*  did and I figured while I was at it I would
*   do all of the conversions for the future.
*     These tables are copied from a book.
*
* If you find any incorrect data or can add some
*     more send them in, and the world can
*            share these routines
*
* If you would like a free update send me a disc
*  and a stamp or request it via e-mail.
*
**************************************************/

#ifndef METRIC__H
#define METRIC__H


/******** Length Conversion Macros **********/

#define CM_TO_IN(x) (0.3937F * (x))
#define IN_TO_CM(x) (2.54F * (x))
#define IN_TO_M(x) (0.0254F * (x))
#define IN_TO_MM(x) (25.4F * (x))
#define KM_TO_MILE(x) (0.6214F * (x))
#define M_TO_IN(x) (39.37F * (x))
#define M_TO_YARD(x) (1.093611F * (x))
#define MILE_TO_KM(x) (1.609F * (x))
#define MM_TO_IN(x) (0.03937F * (x))
#define YARD_TO_M(x) (0.9144F * (x))
#define IN_TO_FT(x) ((x) / 12.0)
#define IN_TO_YARD(x) ((x) / 36.0)
#define IN_TO_MILE(x) ((x) / 63360.0)
#define FT_TO_IN(x) (12.0 * (x))
#define FT_TO_YARD(x) ((x) / 3.0)
#define FT_TO_MILE(x) ((x) / 5280.0)
#define MILE_TO_IN(x) (63360.0 * (x))
#define MILE_TO_FT(x) (5280.0 * (x))
#define MILE_TO_YARD(x) (1760.0 * (x))

/******* Weight Macros ************/

#define G_TO_OZ(x) (0.03527F * (x))
#define KG_TO_LBS(x) (2.2046F * (x))
#define LBS_TO_KG(x) (0.4545F * (x))
#define LBS_TO_OZ(x) (16.0F * (x))
#define OZ_TO_G(x) (28.3527F * (x))
#define OZ_TO_LBS(x) ((x) / 16.0)

/******* Liquid Conversion Macros ********/

#define CL_TO_OZ(x) (0.338F * (x))
#define ML_TO_OZ(x) (0.0338F * (x))
#define OZ_TO_CL(x) (2.95857988F * (x))
#define OZ_TO_ML(x) (29.5857F * (x))

/*********************************************
*
*      In case you are unaware there are:
*           16 ounces in a US quart
*           20 ounces in a UK quart
*
*  This, of course, makes gallons different
*     too.  Supplied below are US to UK
*           conversions as well.
*
*********************************************/

#define KL_TO_UKGAL(x) (211.344F * (x))
#define KL_TO_USGAL(x) (264.18F * (x))
#define L_TO_UKGAL(x) (0.2199F * (x))
#define L_TO_UKQ(x) (0.945F * (x))
#define L_TO_USGAL(x) (0.26455F * (x))
#define L_TO_USQ(x) (1.0582F * (x))
#define UKGAL_TO_KL(x) (0.0047316F * (x))
#define UKGAL_TO_L(x) (4.546F * (x))
#define UKGAL_TO_USGAL(x) (1.25F * (x))
#define UKQ_TO_L(x) (1.0582F * (x))
#define UKQ_TO_USQ(x) (1.25F * (x))
```

```
 101 | #define USGAL_TO_KL(x) (0.003785F * (x))
 102 | #define USGAL_TO_L(x) (3.78F * (x))
 103 | #define USGAL_TO_UKGAL(x) (0.8F * (x))
 104 | #define USQ_TO_L(x) (0.945F * (x))
 105 | #define USQ_TO_UKQ(x) (0.8F * (x))
 106 |
 107 | /**** Area Conversion Macros ****/
 108 |
 109 | #define CM2_TO_IN2(x) (0.155F * (x))
 110 | #define FT2_TO_M2(x) (0.0930F * (x))
 111 | #define IN2_TO_CM2(x) (6.4516F * (x))
 112 | #define IN2_TO_MM2(x) (645.161F * (x))
 113 | #define KM2_TO_MILE2(x) (0.386F * (x))
 114 | #define M2_TO_FT2(x) (10.75F * (x))
 115 | #define MILE2_TO_KM2(x) (2.5906F * (x))
 116 | #define MM2_TO_IN2(x) (0.00155F * (x))
 117 |
 118 | /******** Volume Conversion Macros *********/
 119 |
 120 | #define CM3_TO_IN3(x) (0.061F * (x))
 121 | #define CM3_TO_IN3(x) (0.061F * (x))
 122 | #define IN3_TO_CC(x) (16.3934F * (x))
 123 | #define IN3_TO_CM3(x) (16.3934F * (x))
 124 | #define KM3_TO_MILE3(x) (0.25F * (x))
 125 | #define M3_TO_YARD3(x) (1.308F * (x))
 126 | #define MILE3_TO_KM3(x) (4.0F * (x))
 127 | #define YARD3_TO_M3(x) (0.764526F * (x))
 128 |
 129 | #endif   /* METRIC__H */


TEXT STATISTICS

  4085 characters
   129 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
    64 preprocessor instructions
     0 constants [character]
     0 constants [string]
    61 constants [numeric]

SYMBOL TABLE

    CL_TO_OZ        73
    CM2_TO_IN2             109
    CM3_TO_IN3             120     121
    CM_TO_IN        42
    FT2_TO_M2      110
    FT_TO_IN        55
    FT_TO_MILE              57
    FT_TO_YARD              56
    G_TO_OZ         64
    IN2_TO_CM2             111
    IN2_TO_MM2             112
    IN3_TO_CC      122
    IN3_TO_CM3             123
    IN_TO_CM        43
    IN_TO_FT        52
    IN_TO_M         44
    IN_TO_MILE              54
    IN_TO_MM        45
    IN_TO_YARD              53
    KG_TO_LBS       65
    KL_TO_UKGAL             90
    KL_TO_USGAL             91
    KM2_TO_MILE2           113
    KM3_TO_MILE3           124
    KM_TO_MILE              46
    LBS_TO_KG       66
    LBS_TO_OZ       67
    L_TO_UKGAL              92
    L_TO_UKQ        93
    L_TO_USGAL              94
    L_TO_USQ        95
    M2_TO_FT2      114
    M3_TO_YARD3            125
    METRIC__H       36      37
    MILE2_TO_KM2           115
    MILE3_TO_KM3           126
    MILE_TO_FT              59
    MILE_TO_IN              58
    MILE_TO_KM              49
    MILE_TO_YARD            60
    ML_TO_OZ        74
    MM2_TO_IN2             116
    MM_TO_IN        50
    M_TO_IN         47
    M_TO_YARD       48
    OZ_TO_CL        75
    OZ_TO_G         68
    OZ_TO_LBS       69
    OZ_TO_ML        76
    UKGAL_TO_KL             96
    UKGAL_TO_L              97
    UKGAL_TO_USGAL          98
    UKQ_TO_L        99
```

```
UKQ_TO_USQ              100
USGAL_TO_KL             101
USGAL_TO_L              102
USGAL_TO_UKGAL          103
USQ_TO_L        104
USQ_TO_UKQ              105
YARD3_TO_M3             127
YARD_TO_M        51
x        42+     43+     44+     45+     46+     47+     48+     49+     50+     51+     52+     53+
         54+     55+     56+     57+     58+     59+     60+     64+     65+     66+     67+     68+
         69+     73+     74+     75+     76+     90+     91+     92+     93+     94+     95+     96+
         97+     98+     99+    100+    101+    102+    103+    104+    105+    109+    110+    111+
        112+    113+    114+    115+    116+    120+    121+    122+    123+    124+    125+    126+
        127+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  C macros for min() and max() on any C compiler.
**
**  C++ templates for min() and max() on any C++ compiler which supports
**  templates.
**
**  Overloaded C++ functions for min() and max() on any C++ compiler which
**  doesn't support templates.
**
**  If your compiler supports both C and C++, the appropriate behavior will
**  be selected based on the value of the __cplusplus predefined macro.
*/

#ifndef MINMAX__H
#define MINMAX__H

#undef min
#undef max

/*
**  The following 2 macros are used only for C++ compilers which lack
**  templates. If you use a C++ compiler which supports templates, ignore
**  the rest of this section. If you're compiling standard C, rather than
**  C++, you may ignore the rest of this section.
**
**  Set SIGNED_DISTINCT to 1 for conforming C++ compilers that treat signed,
**  unsigned, and "normal" types as 3 distinct entities. Set SIGNED_DISTINCT
**  to 0 for older or non-conforming C++ compilers. To determine the proper
**  setting for your compiler, compile MINMAX.H as a C++ source file using
**  the "compile only" (usually -c) switch. If the compiler reports "already
**  defined" errors, you probably need to change the value of SIGNED_DISTINCT
**  to 0.
**
**  If you still get "already defined" errors after setting SIGNED_DISTINCT
**  to 0, you may either have a truly broken compiler or have some obscure
**  (e.g. Zortech/Symantec's -Ju or -Jm) switch set which causes this
**  behavior. As a last resort, you can set UNSIGNED_DISTINCT to 0 as well.
*/

#define SIGNED_DISTINCT 1
#define UNSIGNED_DISTINCT 1

/*
**  We can only use templates with newer C++ compilers compiling C++ source
*/

#if defined(__cplusplus) && __cplusplus
 #if  (defined(__SC__) && __SC__ >= 0x700) || \
      (defined(_MSC_VER) && _MSC_VER > 800) || \
      (defined(__WATCOMC__) && __WATCOMC__ >= 1000) || \
      (defined(__BORLANDC__) && __BORLANDC__ >= 0x450)

  template<class T> inline T max(T a, T b) {return (a > b) ? a : b; };
  template<class T> inline T min(T a, T b) {return (a < b) ? a : b; };

 #else /* no templates */

  /*
  ** prototypes for overloaded max() functions
  */

  inline char max(char a, char b)
  {
        return (a > b) ? a : b;
  }
  #if UNSIGNED_DISTINCT
   inline unsigned char max(unsigned char a, unsigned char b)
   {
         return (a > b) ? a : b;
   }
  #endif
  #if SIGNED_DISTINCT
   inline signed char max(signed char a, signed char b)
   {
         return (a > b) ? a : b;
   }
  #endif

  inline short max(short a, short b)
  {
        return (a > b) ? a : b;
  }
  #if UNSIGNED_DISTINCT
   inline unsigned short max(unsigned short a, unsigned short b)
   {
         return (a > b) ? a : b;
   }
  #endif
  #if SIGNED_DISTINCT
   inline signed short max(signed short a, signed short b)
   {
         return (a > b) ? a : b;
   }
  #endif

  inline int max(int a, int b)
  {
        return (a > b) ? a : b;
```

```
101 |   }
102 |   #if UNSIGNED_DISTINCT
103 |    inline unsigned int max(unsigned int a, unsigned int b)
104 |    {
105 |          return (a > b) ? a : b;
106 |    }
107 |   #endif
108 |   #if SIGNED_DISTINCT
109 |    inline signed int max(signed int a, signed int b)
110 |    {
111 |          return (a > b) ? a : b;
112 |    }
113 |   #endif
114 |
115 |   inline long max(long a, long b)
116 |   {
117 |         return (a > b) ? a : b;
118 |   }
119 |   #if UNSIGNED_DISTINCT
120 |    inline unsigned long max(unsigned long a, unsigned long b)
121 |    {
122 |          return (a > b) ? a : b;
123 |    }
124 |   #endif
125 |   #if SIGNED_DISTINCT
126 |    inline signed long max(signed long a, signed long b)
127 |    {
128 |          return (a > b) ? a : b;
129 |    }
130 |   #endif
131 |
132 |   inline float max(float a, float b) {return (a > b) ? a : b; }
133 |
134 |   inline double max(double a, double b) {return (a > b) ? a : b; }
135 |
136 |   /*
137 |   ** prototypes for overloaded min() functions
138 |   */
139 |
140 |   inline char min(char a, char b)
141 |   {
142 |         return (a < b) ? a : b;
143 |   }
144 |   #if UNSIGNED_DISTINCT
145 |    inline unsigned char min(unsigned char a, unsigned char b)
146 |    {
147 |          return (a < b) ? a : b;
148 |    }
149 |   #endif
150 |   #if SIGNED_DISTINCT
151 |    inline signed char min(signed char a, signed char b)
152 |    {
153 |          return (a < b) ? a : b;
154 |    }
155 |   #endif
156 |
157 |   inline short min(short a, short b)
158 |   {
159 |         return (a < b) ? a : b;
160 |   }
161 |   #if UNSIGNED_DISTINCT
162 |    inline unsigned short min(unsigned short a, unsigned short b)
163 |    {
164 |          return (a < b) ? a : b;
165 |    }
166 |   #endif
167 |   #if SIGNED_DISTINCT
168 |    inline signed short min(signed short a, signed short b)
169 |    {
170 |          return (a < b) ? a : b;
171 |    }
172 |   #endif
173 |
174 |   inline int min(int a, int b)
175 |   {
176 |         return (a < b) ? a : b;
177 |   }
178 |   #if UNSIGNED_DISTINCT
179 |    inline unsigned int min(unsigned int a, unsigned int b)
180 |    {
181 |          return (a < b) ? a : b;
182 |    }
183 |   #endif
184 |   #if SIGNED_DISTINCT
185 |    inline signed int min(signed int a, signed int b)
186 |    {
187 |          return (a < b) ? a : b;
188 |    }
189 |   #endif
190 |
191 |   inline long min(long a, long b)
192 |   {
193 |         return (a < b) ? a : b;
194 |   }
195 |   #if UNSIGNED_DISTINCT
196 |    inline unsigned long min(unsigned long a, unsigned long b)
197 |    {
198 |          return (a < b) ? a : b;
199 |    }
200 |   #endif
```

```
201 |   #if SIGNED_DISTINCT
202 |    inline signed long min(signed long a, signed long b)
203 |    {
204 |          return (a < b) ? a : b;
205 |    }
206 |   #endif
207 |
208 |   inline float min(float a, float b) {return (a < b) ? a : b; }
209 |
210 |   inline double min(double a, double b) {return (a < b) ? a : b; }
211 |
212 |  #endif /* no templates */
213 |
214 | #else /* standard C macros */
215 |
216 |  #define min(x,y) (((x) <= (y)) ? (x) : (y))
217 |  #define max(x,y) (((x) >= (y)) ? (x) : (y))
218 |
219 | #endif /* __cplusplus */
220 |
221 | #endif /* MINMAX__H */
```

TEXT STATISTICS

```
  5664 characters
   221 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
    47 preprocessor instructions
     0 constants [character]
     0 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    MINMAX__H       16      17
    SIGNED_DISTINCT         42      74      91      108     125     150     167     184     201
    T       55+     56+
    UNSIGNED_DISTINCT       43      68      85      102     119     144     161     178     195
    _MSC_VER        51+
    __BORLANDC__            53+
    __SC__          50+
    __WATCOMC__             52+
    __cplusplus             49+
    a       55+     56+     64      66+     69      71+     75      77+     81      83+     86      88+
            92      94+     98      100+    103     105+    109     111+    115     117+    120     122+
          126     128+    132+    134+    140     142+    145     147+    151     153+    157     159+
          162     164+    168     170+    174     176+    179     181+    185     187+    191     193+
          196     198+    202     204+    208+    210+
    b       55+     56+     64      66+     69      71+     75      77+     81      83+     86      88+
            92      94+     98      100+    103     105+    109     111+    115     117+    120     122+
          126     128+    132+    134+    140     142+    145     147+    151     153+    157     159+
          162     164+    168     170+    174     176+    179     181+    185     187+    191     193+
          196     198+    202     204+    208+    210+
    defined         49      50      51      52      53
    max             20      55      64      69      75      81      86      92      98      103     109
          115     120     126     132     134     217
    min             19      56      140     145     151     157     162     168     174     179     185
          191     196     202     208     210     216
    x     216+    217+
    y     216+    217+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MKDIRS.C - Function to build multi-level directories in a single call
**
**  Original Copyright 1993-95 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  Also uses PUSHDIR.C from SNIPPETS.
*/

#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <io.h>
#include "dosfiles.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

int mkdirs(char *pathname)
{
      int retval;
      char path[FILENAME_MAX];

      strcpy (path, pathname);            /* isdir() may expand this    */

      while (strlen(path) && '\\' == LAST_CHAR(path))
            LAST_CHAR(path) = NUL;

      while (0 != (retval = mkdir(path)))
      {
            char subpath[FILENAME_MAX] = "", *delim;

            if (EACCES == errno)
            {
                  if (isdir(path))
                        return 0;
                  else  return retval;
            }
            if (NULL == (delim = strrchr(path, '\\')))
                  return retval;
            strncat(subpath, path, delim - path);     /* Appends NUL    */
            if (Success_ != mkdirs(subpath))
                  break;
      }
      return retval;
}

#ifdef TEST

main(int argc, char *argv[])
{
      if (2 > argc)
      {
            puts("Usage: MKDIRS pathname [...pathname]");
            return -1;
      }
      while (--argc)
      {
            ++argv;
            printf("mkdirs(%s) returned %d\n", *argv, mkdirs(*argv));
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1838 characters
    73 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     2 constants [character]
     5 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    EACCES          41
    FILENAME_MAX            30      39
    LAST_CHAR       34      35
    MSDOS           21
    NUL             35
    NULL            47
    Success_        50
    TEST            56
    __MSDOS__       21
    argc            58      60      65
    argv            58      67      68+
    defined         21+
    delim           39      47      49
    errno           18      41
    h       16      17      18      19      24
    io              19
    isdir           43
    main            58
    mkdir           37
    mkdirs          27      50      68
    path            30      32      34+     35      37      43      47      49+
    pathname        27      32
    printf          68
    puts            62
    retval          29      37      45      48      53
    stdlib          16
    strcpy          32
    string          17
    strlen          34
    strncat         49
    strrchr         47
    subpath         39      49      50
    unistd          24
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  MKTONE.C
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stddef.h>
#include "uclock.h"
#include "sound.h"
#include "pchwio.h"


void dosound(int freq)
{
      unsigned i;

      outp(C8253, SETIMER);
      i = (unsigned)freq%256;
      outp(F8253, i);
      i = (unsigned)freq/256;
      outp(F8253, i);
}

void mktone(int freq, int update, unsigned delay)
{
      if (0 == freq)
      {
            soundoff();
            return;
      }
      dosound(freq);
      if (update != UPDATE)
            soundon();
      if (delay == 0)
            return;
      else  usec_delay(1000L * (unsigned long)delay);
      if (update == TOGGLE)
            soundoff();
}

#ifdef TEST

#include <stdio.h>

main()
{
      puts("Playing A2 for 1 sec.");
      mktone(A2, ON, 1000);
      puts("Playing A3 for 1 sec.");
      mktone(A3, UPDATE, 1000);
      mktone(0,  UPDATE, 0);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1282 characters
    62 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     5 constants [string]
    10 constants [numeric]

SYMBOL TABLE

    A2              55
    A3              57
    C8253           24
    F8253           26      28
    ON              55
    SETIMER         24
    TEST            48
    TOGGLE          44
    UPDATE          39      57      58
    delay           31      41      43
    dosound         20      38
    freq            20      25      27      31      33      38
    h       14      50
    i       22      25      26      27      28
    main            52
    mktone          31      55      57      58
    outp            24      26      28
    puts            54      56
    soundoff        35      45
    soundon         40
    stddef          14
    stdio           50
    update          31      39      44
    usec_delay              43
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  MK_FP.H
  5 | **
  6 | **  Standard header file making sure this pesky Intel macro is defined!
  7 | */
  8 |
  9 | #ifndef MK_FP__H
 10 | #define MK_FP__H
 11 |
 12 | #include "extkword.h"
 13 |
 14 | #if defined(__WATCOMC__)
 15 |  #include <i86.h>
 16 | #elif !defined(__PACIFIC__)
 17 |  #include <dos.h>
 18 | #endif
 19 |
 20 | #if !defined(MK_FP)
 21 |     #define MK_FP(seg,off) \
 22 |       ((void FAR *)(((unsigned long)(seg) << 16)|(unsigned)(off)))
 23 | #endif
 24 |
 25 | #endif /* MK_FP__H */
```

TEXT STATISTICS

```
   462 characters
    25 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    FAR             22
    MK_FP           20      21
    MK_FP__H         9      10
    __PACIFIC__             16
    __WATCOMC__             14
    defined         14      16      20
    dos             17
    h       15      17
    i86             15
    off             21      22
    seg             21      22
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** This source is released to the Public
** domain on December 16, 1992.
**
** Curtis Paris
** Internet: cparis@comtch.spk.wa.usa
**
** modified 12-Mar-94 by Bob Stout - Added pc-port.h from SNIPPETS
*/

#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#include "extkword.h"

#define MODEMIO_INIT
#include "modemio.h"

void (INTERRUPT FAR *old_modem_isr)(void);

/*****************************************************************/
void FAR interrupt modem_isr(void)
{
  unsigned char c;

  enable();

  if (modem_buffer_count<1024) {
    c=inp(modem_base);
    if (((c==XON) || (c==XOFF)) && (modem_xon_xoff)) {
      switch(c) {
        case XON :modem_pause=0; break;
        case XOFF:modem_pause=1; break;
      }
    } else {
      modem_pause=0;
      modem_buffer[modem_buffer_head++]=c;
      if (modem_buffer_head>=MAX_BUFFER) modem_buffer_head=0;
      modem_buffer_count++;
    }
    modem_overflow=0;
  } else {
    modem_overflow=1;
  }
  disable();
  outp(0x20,0x20);
}

int com_carrier(void)
{
  int x;

  if (!modem_open) return(0);
  if ((inp(modem_base+6) & 0x80)==128) return(1);

  for (x=0; x<500; x++) {
    if ((inp(modem_base+6) & 0x80)==128) return(1);
  }
  return(0);
}

int com_ch_ready(void)
{
  if (!modem_open) return(0);
  if (modem_buffer_count!=0) return(1);
  return(0);
}

int com_read_ch(void)
/* This returns EOF if the port hasn't been opened, or if no character
 * is waiting.  Otherwise it returns the next character from the port.
 */
{
  unsigned char ch;

  if (!modem_open) return(EOF);
  if (!com_ch_ready()) return(EOF);
  ch=modem_buffer[modem_buffer_tail];
  modem_buffer[modem_buffer_tail]=0;
  modem_buffer_count--;
  if (++modem_buffer_tail>=MAX_BUFFER) modem_buffer_tail=0;
  return(ch);
}

void com_send_ch(unsigned char ch)
{
  if (!modem_open) return;
  outp(modem_base+4,0x0B);
  if (modem_rts_cts) {
    while((inp(modem_base+6) & 0x10)!=0x10) ; /* Wait for Clear to Send */
  }
  while((inp(modem_base+5) & 0x20)!=0x20) ;
  if (modem_xon_xoff) {
    while((modem_pause) && (com_carrier())) ;
  }
  outp(modem_base,ch);
}

```

```
101 |
102 | void com_parity(char p)
103 | {
104 |   int x, newb=0;
105 |
106 |   if (!modem_open) return;
107 |   x=inp(modem_base+3);
108 |
109 |   newb=((x>>6)<<6)+((x<<5)>>5); /* Get rid of old parity */
110 |
111 |   switch(toupper(p)) {
112 |     case 'N':newb+=0x00; break; /* None  */
113 |     case 'O':newb+=0x08;break;  /* Odd   */
114 |     case 'E':newb+=0x18; break; /* Even  */
115 |     case 'M':newb+=0x28;break;  /* Mark  */
116 |     case 'S':newb+=0x38;break;  /* Space */
117 |   }
118 |
119 |   outp(modem_base+3, newb);
120 | }
121 |
122 | void com_data_bits(unsigned char bits)
123 | {
124 |   int x, newb=0;
125 |
126 |   if (!modem_open) return;
127 |   x=inp(modem_base+3);
128 |
129 |     newb=((x>>2)<<2); /* Get rid of the old Data Bits */
130 |
131 |   switch(bits) {
132 |     case 5 :newb+=0x00; break;
133 |     case 6 :newb+=0x01; break;
134 |     case 7 :newb+=0x02; break;
135 |     default:newb+=0x03; break;
136 |   }
137 |
138 |   outp(modem_base+3,newb);
139 | }
140 |
141 | void com_stop_bits(unsigned char bits)
142 | {
143 |   int x, newb=0;
144 |
145 |   if (!modem_open) return;
146 |   x=inp(modem_base+3);
147 |
148 |   newb=((x<<6)>>6)+((x>>5)<<5); /* Kill the old Stop Bits */
149 |
150 |   if (bits==2) newb+=0x04; /* Only check for 2, assume 1 otherwise */
151 |
152 |   outp(modem_base+3,newb);
153 | }
154 |
155 | void com_speed(long speed)
156 | {
157 |   int x;
158 |     char l, m;
159 |     int d;
160 |
161 |   if (!modem_open) return;
162 |
163 |   x=inp(modem_base+3); /* Read In Old Stats */
164 |
165 |   if ((x & 0x80)!=0x80) outp(modem_base+3,x+0x80); /* Set DLab On */
166 |
167 |   d=(int)(115200L/speed);
168 |   l=d & 0xFF;
169 |   m=(d >> 8) & 0xFF;
170 |
171 |   outp(modem_base+0,l);
172 |   outp(modem_base+1,m);
173 |
174 |   outp(modem_base+3,x); /* Restore the DLAB bit */
175 | }
176 |
177 | int com_open(int comport, long speed, int data_bit, unsigned char parity,
178 |       unsigned char stop_bit)
179 | {
180 |   int x;
181 |
182 |   disable();
183 |   if (modem_open) com_close();
184 |   modem_port=comport;
185 |
186 |   switch(modem_port) {
187 |     case 1:modem_base=0x3F8; modem_irq=4; modem_vect=0x0C; break;
188 |     case 2:modem_base=0x2F8; modem_irq=3; modem_vect=0x0B; break;
189 |     case 3:modem_base=0x3E8; modem_irq=4; modem_vect=0x0B; break;
190 |     case 4:modem_base=0x2E8; modem_irq=3; modem_vect=0x0C; break;
191 |     case 5:break;
192 |     default:modem_base=0x3F8; modem_irq=4; modem_vect=0x0C; break;
193 |   }
194 |
195 |   outp(modem_base+1,0x00);
196 |   if (inp(modem_base+1)!=0) {
197 |     enable();
198 |     return(0);
199 |   }
200 |
```

```
201 |   /* Set up the Interrupt Info */
202 |   old_modem_ier=inp(modem_base+1);
203 |   outp(modem_base+1,0x01);
204 |
205 |   old_modem_isr=(void (INTERRUPT FAR *)(void))getvect(modem_vect);
206 |   setvect(modem_vect,modem_isr);
207 |
208 |   if (modem_rts_cts) {
209 |     outp(modem_base+4,0x0B);
210 |   } else {
211 |     outp(modem_base+4,0x09);
212 |   }
213 |
214 |   old_modem_imr=inp(I8088_IMR);
215 |   outp(I8088_IMR,old_modem_imr & ((1 << modem_irq) ^ 0x00FF));
216 |
217 |   for (x=1; x<=5; x++) inp(modem_base+x);
218 |
219 |   modem_open=1;
220 |
221 |   modem_buffer_count=0;
222 |   modem_buffer_head=0;
223 |   modem_buffer_tail=0;
224 |
225 |   com_speed(speed);
226 |   com_data_bits((unsigned char)data_bit);
227 |   com_parity(parity);
228 |   com_stop_bits(stop_bit);
229 |   enable();
230 |   return(1);
231 | }
232 |
233 | void com_close(void)
234 | {
235 |   if (!modem_open) return;
236 |
237 |   outp(modem_base+1,old_modem_ier);
238 |   outp(I8088_IMR, old_modem_imr);
239 |
240 |   setvect(modem_vect, old_modem_isr);
241 |   outp(0x20,0x20);
242 |   modem_open=0;
243 | }
```

```
TEXT STATISTICS

  5317 characters
   243 lines

LEXICAL STATISTICS

    18 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     5 constants [character]
     2 constants [string]
   125 constants [numeric]

SYMBOL TABLE

    EOF             78      79
    FAR             21      24     205
    I8088_IMR      214     215     238
    INTERRUPT       21     205
    MAX_BUFFER              40      83
    MODEMIO_INIT            18
    XOFF            32      35
    XON             32      34
    bits           122     131     141     150
    c       26      31      32+     33      39
    ch              76      80      84      87      98
    com_carrier             51      96
    com_ch_ready            64      79
    com_close      183     233
    com_data_bits          122     226
    com_open       177
    com_parity             102     227
    com_read_ch             71
    com_send_ch             87
    com_speed      155     225
    com_stop_bits          141     228
    comport        177     184
    ctype           15
    d      159     167     168     169
    data_bit       177     226
    disable         47     182
    dos             14
    enable          28     197     229
    getvect        205
    h       13      14      15
    inp             31      56      59      92      94     107     127     146     163     196     202
           214     217
    interrupt       24
    l      158     168     171
    m      158     169     172
    modem_base              31      56      59      90      92      94      98     107     119     127
           138     146     152     163     165     171     172     174     187     188     189     190
           192     195     196     202     203     209     211     217     237
    modem_buffer            39      80      81
    modem_buffer_count              30      41      67      82     221
    modem_buffer_head       39      40+    222
    modem_buffer_tail       80      81      83+    223
    modem_irq      187     188     189     190     192     215
    modem_isr       24     206
    modem_open              55      66      78      89     106     126     145     161     183     219
           235     242
    modem_overflow          43      45
    modem_pause             34      35      38      96
    modem_port             184     186
    modem_rts_cts           91     208
    modem_vect             187     188     189     190     192     205     206     240
    modem_xon_xoff          32      95
    newb           104     109     112     113     114     115     116     119     124     129     132
           133     134     135     138     143     148     150     152
    old_modem_ier          202     237
    old_modem_imr          214     215     238
    old_modem_isr           21     205     240
    outp            48      90      98     119     138     152     165     171     172     174     195
           203     209     211     215     237     238     241
    p      102     111
    parity         177     227
    setvect        206     240
    speed          155     167     177     225
    stdio           13
    stop_bit       178     228
    toupper        111
    x       53      58+    104     107     109+    124     127     129     143     146     148+    157
           163     165+    174     180     217+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*                                          */
/* This source is released to the Public    */
/* domain on December 16, 1992.             */
/*                                          */
/* Curtis Paris                             */
/* Internet: cparis@comtch.spk.wa.usa       */
/*                                          */

#ifndef MODEMIO__H
#define MODEMIO__H

#include "pchwio.h"

#define MAX_BUFFER 1024 /* Max Input Buffer */

#define I8088_IMR  0x21
#define XON        0x11
#define XOFF       0x13

#ifdef MODEMIO_INIT

/*** Modem Buffer Information ***/
int           modem_buffer_count=0;
unsigned char modem_buffer[MAX_BUFFER];
int           modem_buffer_head=0,
              modem_buffer_tail=0;

/*** Misc. Modem Status Information ***/
int modem_rts_cts=1;   /* RTS/CTS variable 0=Off, 1=On */
int modem_xon_xoff=1;  /* XON/XOFF variable 0=off, 1=on */
int modem_open=0;      /* IS the port open variable, do not change */
int modem_port=0;      /* What port is it on, 1-5 */
int modem_base=0;      /* The ports BASE address */
int modem_irq =0;      /* The IRQ */
int modem_vect=0;      /* The Interrupt Vector */
int modem_overflow=0;  /* Modem Overflow Alert */
int modem_pause=0;     /* Is it paused for XON/XOFF */

/*** Old Port Interrupt Holders ***/
int old_modem_lcr, old_modem_imr,
    old_modem_ier, old_modem_status;

#else

/*** Modem Buffer Information ***/
extern int           modem_buffer_count;
extern unsigned char modem_buffer[MAX_BUFFER];
extern int           modem_buffer_head,
                     modem_buffer_tail;

/*** Misc. Modem Status Information ***/
extern int modem_rts_cts;
extern int modem_xon_xoff;
extern int modem_open;
extern int modem_port;
extern int modem_base;
extern int modem_vect;
extern int modem_overflow;
extern int modem_pause;

/*** Old Port Interrupt Holders ***/
extern int old_modem_lcr, old_modem_imr,
       old_modem_ier, old_modem_status;

#endif

int com_carrier(void);

int  com_ch_ready(void);
int  com_read_ch(void);
void com_send_ch(unsigned char ch);

void com_parity(char p);
void com_data_bits(unsigned char bits);
void com_stop_bits(unsigned char bits);
void com_speed(long speed);

int com_open(int comport, long speed, int data_bit, unsigned char parity,
      unsigned char stop_bit);
void com_close(void);

#endif /* MODEMIO__H */
```

```
TEXT STATISTICS

  2454 characters
    84 lines

LEXICAL STATISTICS

    25 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     1 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    I8088_IMR       18
    MAX_BUFFER              16      26      49
    MODEMIO_INIT            22
    MODEMIO__H              11      12
    XOFF            20
    XON             19
    bits            76      77
    ch              73
    com_carrier             69
    com_ch_ready            71
    com_close       82
    com_data_bits           76
    com_open        80
    com_parity              75
    com_read_ch             72
    com_send_ch             73
    com_speed       78
    com_stop_bits           77
    comport         80
    data_bit        80
    modem_base              35      58
    modem_buffer            26      49
    modem_buffer_count              25      48
    modem_buffer_head       27      50
    modem_buffer_tail       28      51
    modem_irq       36
    modem_open              33      56
    modem_overflow          38      60
    modem_pause             39      61
    modem_port              34      57
    modem_rts_cts           31      54
    modem_vect              37      59
    modem_xon_xoff          32      55
    old_modem_ier           43      65
    old_modem_imr           42      64
    old_modem_lcr           42      64
    old_modem_status        43      65
    p       75
    parity          80
    speed           78      80
    stop_bit        81
```

```
/* +++Date last modified: 05-Jul-1997 */

/**********************************************************************
* @(#) MODULINF.h
* @(#)      Header defining a structure for tracking & status information.
*
*
*** Corrections ***************************************************
*
*
**********************************************************************
*@(#)1993 Erik Bachmann (E-mail: ebp@dde.dk)
*
* Released to public domain 27-Oct-95
**********************************************************************/

#if !defined(MODULINF_H)      /* IF not yet defined (to avoid repetitions) */
#define MODULINF_H                                        /* Define */

struct tyModulInfo
{
      char  *pszModuleTag ;                         /* tagModulInf    */
      char  *pszModuleName ;                        /* __FILE__       */
      char  *pszCreateDate ;                        /* 1993-06-11     */
      char  *pszChangedDate ;                       /* __DATE__       */
      char  *pszChangedTime ;                       /* __TIME__       */
      char  *pszCopyright ;                         /* text           */
} ;

#endif


#define MODULEINF( _MODULEDATE, _MODULENAME ) \
      static struct tyModulInfo stModulInfo = \
      { \
            "tagModulInf", \
            __FILE__,\
             _MODULEDATE,\
             __DATE__,\
            __TIME__, \
            _MODULENAME \
      }

#define PROGRAMINF( _PROGNAME, _PROGVER, _MODULEDATE, _COPYRIGHT ) \
      const char  far   *PROGNAME   =     _PROGNAME ; \
      const char  far   *PROGVER          =     _PROGVER ; \
      struct tyModulInfo stModulInfo = \
      { \
            "tagModulInf",\
             __FILE__,\
            _MODULEDATE,\
             __DATE__,\
            __TIME__, \
            _COPYRIGHT \
      }
```

```
TEXT STATISTICS

  1898 characters
    55 lines

LEXICAL STATISTICS

    10 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    MODULEINF       33
    MODULINF_H              17      18
    PROGNAME        45
    PROGRAMINF              44
    PROGVER         46
    _COPYRIGHT              44      54
    _MODULEDATE             33      38      44      51
    _MODULENAME             33      41
    _PROGNAME       44      45
    _PROGVER        44      46
    __DATE__        39      52
    __FILE__        37      50
    __TIME__        40      53
    defined         17
    far             45      46
    pszChangedDate          25
    pszChangedTime          26
    pszCopyright            27
    pszCreateDate           24
    pszModuleName           23
    pszModuleTag            22
    stModulInfo             34      47
    tyModulInfo             20      34      47
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)MODULUS.c
* @(#)- Basic routines for creating and checking modulus values
*
*
*****************************************************************************
* CONTENTS:
*
*   modulus10() ;        Returns the modulus 10 check digit.
*   modulus11() ;        Returns the modulus 11 check digit
*   check_modulus10() ;  Checks if the check digit in string is correct
*   check_modulus11() ;  Checks if the check digit in string is correct
*
*****************************************************************************
* DESCRIPTION:
*
* Calculations of modulus 10 check digits:
*
*   Multiply units position an every       Base 6 1 2 4 8
*   alternate position i basic code             6   2   8
*   number by 2, starting from righthand
*   digit                                                 = 1256
*
*   Bring down digits that were not              1   4
*   multiplied by 2
*
*   Cross add these digits to digits in
*   first product                             1+2+5+6+1+4 = 19
*
*   Enter next higher number with a ending
*   zero. If the number already ends with
*   zero keep it.                                          20
*
*   Subtract the cross add sum with next
*   higher and you've got the check digit         20 - 19 = 1
*                                                           Í
*   Result : 61248 1
*
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* Calculations of modulus 11 check digits:
*
*   Assigned weighted checking factor    base       9 4 3 4 5 7 8 4 2
*   to each digits position of basic     weight     4 3 2 7 6 5 4 3 2
*   code number starting with units
*   position of number and progressing
*   to most significant digit
*
*   Weight used is:
*   765432765432765432765432765432765432 etc.
*
*   Multiply esch digit of the basic               36
*   code number by its weighted checking             12
*   factor, and add the products                        6
*                                                        28
*                                                          30
*                                                            35
*                                                              32
*                                                                12
*                                                                   4
*
*
*                                                          Sum = 195
*
*   Divide Sum of products by 11                  195 % 11 = 17 plus
remainder of 8
*
*   Subtract remainder from 11                    11 - 8 = 3
*   difference is check digit                             Í
*
*   Result :  943457842 3
*
*****************************************************************************
*@(#)1993 Erik Bachmann (E-mail: ebp@dde.dk)
*
* Released to public domain 27-Oct-95
*****************************************************************************/
#include <math.h>                     /* pow() */
#include <string.h>                   /* strlen(), strncpy() */
#include "bacstd.h"


/*
 /-------------------------------------\
 |      MODULUS10                      |-----------------------------------|
 |\-------------------------------------/
 |
 |
 |-------------------------------------------------------------------------|
 | CALL:
 |    iMod10 = modulus10("61248") ;
 |
 | HEADER:
 |    math.h        pow()
 |
 | GLOBALE VARIABLES:
 |    %
```

```
101 | |
102 | | ARGUMENTS:
103 | |    char *pszBase
104 | |                  The string on which the modulus is performed
105 | |
106 | |
107 | | PROTOTYPE:
108 | |    int _CfnTYPE modulus10(char *pszBase) ;
109 | |
110 | | RETURN VALUE:
111 | |    int           Modulus value
112 | |
113 | | MODULE:
114 | |    modulus.c
115 | |-------------------------------------------------------------------------|
116 | |
117 | |
118 | |-------------------------------------------------------------------------|
119 | |1993-09-29/Erik Bachmann
120 | \-------------------------------------------------------------------------|*/
121 |
122 | int _CfnTYPE modulus10(char *pszBase)
123 | {
124 |    int   iSum  = 0,
125 |          iTemp = 0,
126 |          i,
127 |          j ;
128 |
129 |    /*--------------------------------*/
130 |
131 |                                      /* Get units position an every
132 |                                         alternate position i basic code,
133 |                                         starting from righthand and
134 |                                         cross add */
135 |    for (i = strlen(pszBase) - 1, j = 0 ; i >= 0 ; i -= 2, j++ )
136 |    {
137 |       iSum += (pszBase[i] - '0') * (int)pow(10, j) ;
138 |    }
139 |
140 |    iSum *= 2 ;                       /* Multiply by 2 */
141 |
142 |    iTemp = iSum ;
143 |                                      /* Bring down digits that were not
144 |                                         multiplied by 2 */
145 |    for ( iSum = 0 ; j >= 0 ; j--)
146 |    {
147 |       iSum  += iTemp % 10 ;             /* Cross add these digits to digits in
148 |                                         first product */
149 |       iTemp /= 10 ;
150 |    }
151 |                                      /* Enter next higher number with a
152 |                                         ending zero. If the number already
153 |                                         ends with zero keep it. */
154 |
155 |    for (i = strlen(pszBase) - 2 ; i >= 0 ; i -= 2 )
156 |    {
157 |       iSum += (pszBase[i] - '0') ;
158 |    }
159 |
160 |                                      /* Subtract the cross add sum with
161 |                                         next higher and you've got the
162 |                                         check digit*/
163 |    if ( (iSum % 10) != 0)
164 |    {
165 |       iSum = ( ( (iSum / 10) + 1 ) * 10) - iSum ;
166 |    }
167 |    else
168 |       iSum = 0 ;
169 |
170 |    return( iSum ) ;
171 | }  /*** modulus10() ***/
172 |
173 |
174 | /*
175 |  /-------------------------------------\
176 | |      MODULUS11                        |-----------------------------------|
177 | |\-------------------------------------/
178 | |
179 | |
180 | |-------------------------------------------------------------------------|
181 | | CALL:
182 | |    iMod11 = modulus11("943457842") ;
183 | |
184 | | HEADER:
185 | |    math.h        pow()
186 | |
187 | | GLOBALE VARIABLES:
188 | |    %
189 | |
190 | | ARGUMENTS:
191 | |    char *pszBase
192 | |                  The string on which the modulus is performed
193 | |
194 | | PROTOTYPE:
195 | |    int _CfnTYPE modulus11(char *pszBase) ;
196 | |
197 | | RETURN VALUE:
198 | |    int           Modulus value
199 | |
200 | | MODULE:
```

```
201 | |    modulus.c
202 | |-------------------------------------------------------------------------|
203 | |
204 | |
205 | |-------------------------------------------------------------------------|
206 | |1993-09-29/Erik Bachmann
207 | \-------------------------------------------------------------------------|*/
208 |
209 | int _CfnTYPE modulus11(char *pszBase)
210 | {
211 |    char *pszWeids = "765432" ;
212 |    int   iSum = 0 ;
213 |    int   i,
214 |          j ;
215 |
216 |    /*--------------------------*/
217 |
218 |                                      /* Assigned weighted checking factor
219 |                                         to each digits position of basic
220 |                                         code number starting with units
221 |                                         position of number and progressing
222 |                                         to most significant digit */
223 |
224 |    for ( i = strlen(pszBase) - 1, j = 5 ; i >= 0 ; i-- )
225 |    {
226 |                                      /* Multiply esch digit of the basic
227 |                                         code number by its weighted checking
228 |                                         factor, and add the products */
229 |       iSum += (pszBase[i] - '0') * (pszWeids[j--] - '0') ;
230 |       if (j < 0)
231 |          j = 5 ;
232 |    }
233 |
234 |                                      /* Divide Sum of products by 11
235 |                                         Subtract remainder from 11
236 |                                         difference is check digit */
237 |    iSum = 11 - (iSum % 11) ;
238 |
239 |    return( iSum ) ;
240 | }  /*** modulus11() ***/
241 |
242 |
243 | /*-------------------------------------------------------------*/
244 |
245 | /*
246 |  /-----------------------------------\
247 | |      CHECK_MODULUS10                |-----------------------------------|
248 | |\----------------------------------/
249 | |
250 | |
251 | |-------------------------------------------------------------------------|
252 | | CALL:
253 | |    iMod10 = check_modulus10("612481") ;
254 | |
255 | | HEADER:
256 | |    math.h        pow()
257 | |
258 | | GLOBALE VARIABLES:
259 | |    %
260 | |
261 | | ARGUMENTS:
262 | |    char *pszBase
263 | |                  The string including the digit to be checked
264 | |
265 | |
266 | | PROTOTYPE:
267 | |    int _CfnTYPE check_modulus10(char *pszBase) ;
268 | |
269 | | RETURN VALUE:
270 | |    int           0     = OK
271 | |                  else    value - 10 = correct digit
272 | | MODULE:
273 | |    modulus.c
274 | |-------------------------------------------------------------------------|
275 | |
276 | |
277 | |-------------------------------------------------------------------------|
278 | |1993-09-29/Erik Bachmann
279 | \-------------------------------------------------------------------------|*/
280 |
281 | int _CfnTYPE check_modulus10(char *pszBase)
282 | {
283 |    char  szTmp[11] ;
284 |    int   iTmp = 0 ;
285 |
286 |    /*--------------------*/
287 |
288 |    strncpy(szTmp, pszBase, strlen(pszBase) - 1) ;
289 |    szTmp[strlen(pszBase) - 1] = '\0' ;
290 |                                      /* Extract base */
291 |    iTmp = pszBase[strlen(pszBase) - 1] - '0' ;
292 |                                      /* Extract current check digit */
293 |    if ( modulus10(szTmp) == iTmp )
294 |       return( 0 ) ;                  /* Check digit is OK */
295 |    else
296 |       return( 10 + modulus10(szTmp) ) ;/* On error return the correct + 10 */
297 | }  /*** check_modulus10() ***/
298 |
299 | /*-------------------------------------------------------------*/
300 |
```

```
/*
 /-------------------------------------\
|      CHECK_MODULUS11                  |-----------------------------------|
|\-------------------------------------/
|
|
|-------------------------------------------------------------------------------|
| CALL:
|    iMod11 = check_modulus11("9434578423") ;
|
| HEADER:
|    math.h        pow()
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    char *pszBase
|                  The string including the digit to be checked
|
|
| PROTOTYPE:
|    int _CfnTYPE check_modulus10(char *pszBase) ;
|
| RETURN VALUE:
|    int           0     = OK
|                  else    value - 11 = correct digit
| MODULE:
|    modulus.c
|-------------------------------------------------------------------------------|
|
|
|-------------------------------------------------------------------------------|
|1993-09-29/Erik Bachmann
\-------------------------------------------------------------------------------|*/

int _CfnTYPE check_modulus11(char *pszBase)
{
   char  szTmp[11] ;
   int   iTmp = 0 ;

   /*--------------------*/

   strncpy(szTmp, pszBase, strlen(pszBase) - 1) ;
   szTmp[strlen(pszBase) - 1] = '\0' ;
                                    /* Extract base */
   iTmp = pszBase[strlen(pszBase) - 1] - '0' ;
                                    /* Extract current check digit */

   if ( modulus11(szTmp) == iTmp )
      return( 0 ) ;                 /* Check digit is OK */
   else
      return( 11 + modulus11(szTmp) ) ;
                                    /* On error return the correct + 11 */
}  /*** check_modulus11() ***/


/************************************************/

#ifdef   TEST

int main(void)
{
   int iStatus ;

   /*------------------------------*/

   iStatus = modulus11("943457842") ;
   iStatus = check_modulus11("9434578423") ;
   iStatus = modulus10("61248") ;
   iStatus = check_modulus10("612481") ;

   return( iStatus ) ;
}
#endif
```

```
TEXT STATISTICS

 18496 characters
   375 lines

LEXICAL STATISTICS

    37 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     8 constants [character]
     6 constants [string]
    43 constants [numeric]

SYMBOL TABLE

    TEST           360
    _CfnTYPE       122     209     281     337
    check_modulus10        281     371
    check_modulus11        337     369
    h       81      82
    i      126     135+    137     155+    157     213     224+    229
    iStatus        364     368     369     370     371     373
    iSum           124     137     140     142     145     147     157     163     165+    168     170
           212     229     237+    239
    iTemp          125     142     147     149
    iTmp           284     291     293     340     347     350
    j      127     135+    137     145+    214     224     229     230     231
    main           362
    math            81
    modulus10      122     293     296     370
    modulus11      209     350     353     368
    pow            137
    pszBase        122     135     137     155     157     209     224     229     281     288+    289
          291+     337     344+    345     347+
    pszWeids       211     229
    string          82
    strlen         135     155     224     288     289     291     344     345     347
    strncpy        288     344
    szTmp          283     288     289     293     296     339     344     345     350     353
```

```
/* +++Date last modified: 05-Jul-1997 */

/* PD by Michelangelo Jones, 1:1/124. */

/*
**  Returns 0 for new moon, 15 for full moon,
**  29 for the day before new, and so forth.
*/

/*
**  This routine sometimes gets "off" by a few days,
**  but is self-correcting.
*/

#include "datetime.h"

int moon_age(int month, int day, int year)
{
      static short int ages[] =
            {18, 0, 11, 22, 3, 14, 25, 6, 17,
             28, 9, 20, 1, 12, 23, 4, 15, 26, 7};
      static short int offsets[] =
            {-1, 1, 0, 1, 2, 3, 4, 5, 7, 7, 9, 9};

      if (day == 31)
            day = 1;
      return ((ages[(year + 1) % 19] + ((day + offsets[month-1]) % 30) +
            (year < 1900)) % 30);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

static char *description[] = {
      "new",                  /* totally dark                        */
      "waxing crescent",      /* increasing to full & quarter light  */
      "in its first quarter", /* increasing to full & half light     */
      "waxing gibbous",       /* increasing to full & > than half    */
      "full",                 /* fully lighted                       */
      "waning gibbous",       /* decreasing from full & > than half  */
      "in its last quarter",  /* decreasing from full & half light   */
      "waning crescent"       /* decreasing from full & quarter light */
      };

static char *months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

int main(int argc, char *argv[])
{
      int month, day, year, phase;

      if (4 > argc)
      {
            puts("Usage: MOON_AGE month day year");
            return EXIT_FAILURE;
      }
      month = atoi(argv[1]);
      day   = atoi(argv[2]);
      year  = atoi(argv[3]);
      if (100 > year)
            year += 1900;
      printf("moon_age(%d, %d, %d) returned %d\n", month, day, year,
            phase = moon_age(month, day, year));
      printf("Moon phase on %d %s %d is %s\n", day, months[month - 1], year,
            description[(int)((phase + 2) * 16L / 59L)]);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2218 characters
    71 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
    24 constants [string]
    49 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            57
    EXIT_SUCCESS            68
    TEST            31
    ages            19      27
    argc            50      54
    argv            50      59      60      61
    atoi            59      60      61
    day             17      25      26      27      52      60      64      65      66
    description             36      67
    h       33      34
    main            50
    month           17      27      52      59      64      65      66
    months          47      66
    moon_age        17      65
    offsets         22      27
    phase           52      65      67
    printf          64      66
    puts            56
    stdio           33
    stdlib          34
    year            17      27      28      52      61      62      63      64      65      66
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  MORE.C - A DOS/Unix more work-alike
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include "more.h"
 11 |
 12 | main()
 13 | {
 14 |       char buf[512];
 15 |
 16 |       while(NULL != fgets(buf, 512, stdin))
 17 |       {
 18 |             if (Key_ESC == more_proc(buf))
 19 |                   break;
 20 |       }
 21 |       return 0;
 22 | }
```

TEXT STATISTICS

```
   360 characters
    22 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    Key_ESC         18
    NULL            16
    buf             14      16      18
    fgets           16
    h        9
    main            12
    more_proc       18
    stdin           16
    stdio            9
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPPETS header file for MOREPROC.C & ADJSCROL.C
  5 | **
  6 | **  Note: As written, this code is DOS-specific. To enhance portability:
  7 | **
  8 | **        1. Use UNXGETCH.C from SNIPPETS for portable EXT_KEYS.H functions.
  9 | **        2. Use constants or other non-DOS definitions for SCRNxxxx macros.
 10 | **        3. Conditional code in DELAY.C allows use with non-DOS compilers.
 11 | */
 12 |
 13 | #ifndef MORE__H
 14 | #define MORE__H
 15 |
 16 | #include "scrnmacs.h"                    /* For SCREENROWS, SCREENCOLS */
 17 | #include "snip_str.h"                    /* For strn1cpy(), strmove()  */
 18 | #include "ext_keys.h"                    /* For ext_getch()            */
 19 | #include "delay.h"                       /* For delay()                */
 20 |
 21 | int more_proc(char *str);                           /* MOREPROC.C     */
 22 | int adj_scroll(char *str);                          /* ADJSCROL.C     */
 23 |
 24 | #endif /* MORE__H */
```

TEXT STATISTICS

```
   941 characters
    24 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     4 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    MORE__H         13      14
    adj_scroll              22
    more_proc       21
    str             21      22
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  MOREPROC.C - Display lines to the screen using more-style processing
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include "more.h"
 11 |
 12 | static int rows, cols;
 13 |
 14 | /*
 15 | **  more_proc() - Display a line using more-style processing
 16 | **
 17 | **  Parameters: 1 - Line to display
 18 | **
 19 | **  Returns: Key_ESC ('\x1b') if ESC pressed, else 0
 20 | **
 21 | **  Notes: Reads screen size from BIOS
 22 | **         Handles long line wrapping
 23 | **         Space to single step
 24 | **         ESC to exit via calling function
 25 | **         Any other key to continue until screen full
 26 | */
 27 |
 28 | int more_proc(char *str)
 29 | {
 30 |       static int linecnt = 0;
 31 |       char linebuf[256];
 32 |
 33 |       if (!rows || !cols)
 34 |       {
 35 |             rows = SCREENROWS;
 36 |             cols = SCREENCOLS;
 37 |       }
 38 |
 39 |       if (strlen(str) == ((size_t)cols + 1) && LAST_CHAR(str) == '\n')
 40 |                   LAST_CHAR(str) = NUL;
 41 |
 42 |       while (strlen(str) > (size_t)cols)
 43 |       {
 44 |             strn1cpy(linebuf, str, cols);
 45 |             linebuf[cols] = NUL;
 46 |             more_proc(linebuf);
 47 |             strMove(str, str + cols);
 48 |       }
 49 |
 50 |       if ((rows - (++linecnt)) < 2)
 51 |       {
 52 |             int ch;
 53 |
 54 |             fputs("[...more...]", stderr);
 55 |             fflush(stderr);
 56 |
 57 |             switch (ch = ext_getch())
 58 |             {
 59 |             case ' ':
 60 |                   --linecnt;
 61 |                   break;
 62 |
 63 |             case Key_ESC:
 64 |                   fputs("\r[...Aborted...]\n", stderr);
 65 |                   return ch;
 66 |
 67 |             default:
 68 |                   linecnt = 0;
 69 |             }
 70 |             fputs("\r            \r", stderr);
 71 |             fflush(stderr);
 72 |       }
 73 |
 74 |       fputs(str, stderr);
 75 |       return 0;
 76 | }
 77 |
 78 | #ifdef TEST
 79 |
 80 | #include "errors.h"                           /* For cant()          */
 81 |
 82 | main(int argc, char *argv[])
 83 | {
 84 |       FILE *fp;
 85 |       char buf[512];
 86 |
 87 |       while (--argc)
 88 |       {
 89 |             fp = cant(*++argv, "r");
 90 |
 91 |             while (!feof(fp))
 92 |             {
 93 |                   if (NULL == fgets(buf,  512, fp))
 94 |                         break;
 95 |                   if (Key_ESC == more_proc(buf))
 96 |                         break;
 97 |             }
 98 |       }
 99 |       return 0;
100 | }
```

```
 101 |
 102 | #endif /* TEST */


TEXT STATISTICS

  2266 characters
   102 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     2 constants [character]
     6 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    FILE            84
    Key_ESC         63      95
    LAST_CHAR       39      40
    NUL             40      45
    NULL            93
    SCREENCOLS              36
    SCREENROWS              35
    TEST            78
    argc            82      87
    argv            82      89
    buf             85      93      95
    cant            89
    ch              52      57      65
    cols            12      33      36      39      42      44      45      47
    ext_getch       57
    feof            91
    fflush          55      71
    fgets           93
    fp              84      89      91      93
    fputs           54      64      70      74
    h        9
    linebuf         31      44      45      46
    linecnt         30      50      60      68
    main            82
    more_proc       28      46      95
    rows            12      33      35      50
    size_t          39      42
    stderr          54      55      64      70      71      74
    stdio            9
    str             28      39+     40      42      44      47+     74
    strMove         47
    strlen          39      42
    strn1cpy        44
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**   <<< Morse Code Functions >>>
**
** Written by Michael M. Dodd, N4CF, and placed in the public domain.
**
** The morse() function transmits a string in Morse code on the IBM PC's
** speaker.  The speed is set by a program constant (UNIT_TIME).
**
** There are several other functions in this file, all used by morse(),
** and defined ahead of morse() for convenience.
**
** The main() function at the end of the file is a test program to send
** the command-line argument string in morse code.  Enclose multiple
** words in quotes.  Example:  morse "hello, world"
**
** These functions have been compiled and tested in the Small and Large
** memory models using Microsoft C 6.00a.
**
** Modified for ZTC++, TC++, & BC++ by Bob Stout
*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <ctype.h>

/*
** These functions turn on and off the CW tone on the PC's speaker.  The
** frequency is specified by the freq argument.
** IMPORTANT!  These functions are highly IBM PC-specific!
*/

#define CLK_FREQ (1193180L)
#define PIO (0x61)
#define CTC_CMD (0x43)
#define CTC_DATA (0x42)
#define SETUP (0xB6)
#define TONE_ON (0x03)
#define TONE_OFF (0xFC)

void note_on (int freq)        /* Turn on the tone.  */
{
      int divisor ;
      int pio_word ;

      divisor = (int)(CLK_FREQ / (long)(freq)) ;
      outp (CTC_CMD, SETUP) ;
      outp (CTC_DATA, divisor & 0xFF) ;
      outp (CTC_DATA, divisor >> 8) ;
      pio_word = inp (PIO) ;
      outp (PIO, pio_word | TONE_ON) ;
}

void note_off (void)           /* Turn off the tone.  */
{
      int pio_word ;

      pio_word = inp (PIO) ;
      outp (PIO, pio_word & TONE_OFF) ;
}

/*
** These functions implement a timing-loop delay.  Because the PC's
** internal clock is too coarse for accurate CW timing, the pause()
** function uses a simple software loop to produce a delay.
**
** To minimize the effects of CPU clock speed, the calib() function
** returns a number which represents a rough index of the clock speed
** with relation to the standard IBM PC (this is very approximate).
**
** Calibration is performed only once, when the static fudge_factor is
** zero.  Thereafter, the contents of fudge_factor are used to form a
** delay value.
**
** IMPORTANT!  These functions are highly IBM PC-specific!
*/

unsigned int calib (void)
{
      unsigned int far *timerLow = (unsigned int far *)(0x046c) ;
      unsigned int lastTime ;
      unsigned int iter ;

      for (lastTime = *timerLow; lastTime == *timerLow;)
            ;

      for (iter = 0, lastTime = *timerLow; lastTime == *timerLow; iter++)
            ;
#if   defined(__ZTC__)
      return ((unsigned int)((125L * ((long)(iter)) + 50L) / 2300L)) ;
#elif defined(__TURBOC__)
      return ((unsigned int)((77L * ((long)(iter)) + 50L) / 2300L)) ;
#else /* assume MSC */
      return ((unsigned int)((100L * ((long)(iter)) + 50L) / 2300L)) ;
#endif
}

void pause (unsigned int amount)
```

```
{
      static unsigned int fudge_factor = 0 ;
      unsigned long ul ;

      if (fudge_factor == 0)       /* Calibrate the speed.  */
            fudge_factor = calib () ;

      ul = (unsigned long)(amount) * (long)(fudge_factor) ;
      while (ul--)                 /* Delay.  */
            ;
}

/*
** These functions transmit a dot, a dash, a letter space, and a
** word space.
**
** Note that a single unit space is automatically transmitted after
** each dot or dash, so the ltr_space() function produces only a
** two-unit pause.
**
** Also, the word_space() function produces only a four-unit pause
** because the three-unit letter space has already occurred following
** the previous letter.
*/

#define SPACE_MASK (1 << 15)
#define BIT_MASK (0xfe)
#define UNIT_TIME (18)
#define FREQUENCY (1500)

void send_dot (void)           /* Send a dot and a space.  */
{
      note_on (FREQUENCY) ;
      pause (UNIT_TIME) ;
      note_off () ;
      pause (UNIT_TIME) ;
}

void send_dash (void)          /* Send a dash and a space.  */
{
      note_on (FREQUENCY) ;
      pause (UNIT_TIME * 3) ;
      note_off () ;
      pause (UNIT_TIME) ;
}

void ltr_space (void)          /* Produce a letter space.  */
{
      pause (UNIT_TIME * 2) ;
}

void word_space (void)         /* Produce a word space.  */
{
      pause (UNIT_TIME * 4) ;
}


/*
** MORSE () - Transmit a string in Morse code
**
** This function transmits the string pointed to by the cp argument in
** Morse code on the PC's speaker.  The speed is set by the UNIT_TIME
** constant.
**
** A static table translates from ASCII to Morse code.  Each entry is
** an unsigned integer, where a zero represents a dot and a one
** represents a dash.  No more than 14 bits may be used.  Setting bit
** 15 produces a word space, regardless of any other pattern.
**
** The Morse code pattern is taken from bit 0, and is shifted right
** each time an element is sent.  A special "marker bit" follows the
** complete Morse pattern.  This marker bit is tested before
** transmitting each bit; if there are no 1's in bits 1..15, the
** complete character has been sent.
**
** For example, an "L" would be 0000000000010010, with bit zero
** containing the first dot, bit one the dash, etc.  The marker
** bit is in bit 4.
*/

void morse (char *cp)
{  /*--- MORSE CODE FUNCTION ---*/

      unsigned int c ;
      static unsigned int codes [64] = {
            SPACE_MASK,                    /* Entry 0 = space (0x20)  */
            0, 0, 0, 0, 0, 0, 0, 0,        /* ! " # $  % & " (   */
            0, 0, 0, 115, 49, 106, 41,     /* ) * + , - . /      */
            63, 62, 60, 56, 48, 32, 33, 35,  /* 0 1 2 3 4 5 6 7   */
            39, 47, 0, 0, 0, 0, 0, 76,     /* 8 9 : ; < = > ?   */
            0, 6, 17, 21, 9, 2, 20, 11,    /* @ A B C D E F G   */
            16, 4, 30, 13, 18, 7, 5, 15,   /* H I J K L M N O   */
            22, 27, 10, 8, 3, 12, 24, 14,  /* P Q R S T U V W   */
            25, 29, 19                     /* X Y Z  */
            } ;

      pause (0) ;                /* Calibrate pause() function.  */

      while ((c = *cp++) != '\0')
      {  /*--- TRANSMIT COMPLETE STRING ---*/
```

```
201 |
202 |             c = toupper (c) ;          /* No lower-case Morse characters.  */
203 |             c -= ' ' ;                 /* Adjust for zero-based table.  */
204 |
205 |             if (c > 58)                /* If out of range, ignore it.  */
206 |                   continue ;
207 |
208 |             c = codes[c] ;             /* Look up Morse pattern from table. */
209 |
210 |             if (c & SPACE_MASK)        /* If the space bit is set..  */
211 |             {                          /* ..send a word space and go on.  */
212 |                   word_space () ;
213 |                   continue ;
214 |             }
215 |
216 |             while (c & BIT_MASK)       /* Transmit one character.  */
217 |             {  /*--- TRANSMIT EACH BIT ---*/
218 |                   if (c & 1)
219 |                         send_dash () ;
220 |                   else  send_dot () ;
221 |
222 |                   c >>= 1 ;
223 |             }  /*--- TRANSMIT EACH BIT ---*/
224 |
225 |             ltr_space () ;             /* Send a space following character. */
226 |
227 |       }  /*--- TRANSMIT COMPLETE STRING ---*/
228 |
229 | }  /*--- MORSE CODE FUNCTION ---*/
230 |
231 |
232 | /*
233 | ** This is the test program, which transmits argv[1] in Morse code.
234 | */
235 |
236 | main (int argc, char *argv[])
237 | {
238 |       while (--argc)
239 |             morse (*++argv) ;
240 |       return 0;
241 | }
```

TEXT STATISTICS

```
  7601 characters
   241 lines
```

LEXICAL STATISTICS

```
    40 comments [std-C]
     0 comments [C++]
    19 preprocessor instructions
     2 constants [character]
     0 constants [string]
    94 constants [numeric]
```

SYMBOL TABLE

```
    BIT_MASK       127     216
    CLK_FREQ        35      48
    CTC_CMD         37      49
    CTC_DATA        38      50      51
    FREQUENCY      129     133     141
    PIO             36      52      53      60      61
    SETUP           39      49
    SPACE_MASK             126     186     210
    TONE_OFF        41      61
    TONE_ON         40      53
    UNIT_TIME      128     134     136     142     144     149     154
    __TURBOC__              93
    __ZTC__         91
    amount         100     108
    argc           236     238
    argv           236     239
    c      184     199     202+    203     205     208+    210     216     218     222
    calib           80     106
    codes          185     208
    conio           26
    cp             181     199
    ctype           27
    defined         91      93
    divisor         45      48      50      51
    dos             25
    far             82+
    freq            43      48
    fudge_factor           102     105     106     108
    h       24      25      26      27
    inp             52      60
    iter            84      89+     92      94      96
    lastTime        83      86+     89+
    ltr_space      147     225
    main           236
    morse          181     239
    note_off        56     135     143
    note_on         43     133     141
    outp            49      50      51      53      61
    pause          100     134     136     142     144     149     154     197
    pio_word        46      52      53      58      60      61
    send_dash      139     219
    send_dot       131     220
    stdio           24
    timerLow        82      86+     89+
    toupper        202
    ul             103     108     109
    word_space             152     212
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** A series of routines to provide access to MicroSoft (and compatible)
** mice.  Consult your mouse documentation for detailed information regarding
** each mouse driver function.
**
** by Bob Jarvis w/ modifications by Bob Stout & Bruce Wedding
*/

#include <dos.h>
#include "mouse.h"

int mouse_present = 0;  /* globally visible */

#define DOS_INT 0x21

#define MOUSE(workregs) int86(MSMOUSE,&workregs,&workregs)
#define MOUSEX(workregs,sregs) int86x(DOS_INT,&workregs,&workregs,&sregs)

/*
** Uses driver function 0 to initialize the mouse software to its default
** settings.  If no mouse is present it returns 0.  If a mouse is present, it
** returns -1, and places the value of the mouse type (2 = MicroSoft,
** 3 = Mouse Systems, other values are possible) in *mousetype.  Also
** initializes the global variable mouse_present (0 = no mouse, !0 = mouse
** is available).
*/

int ms_reset(int *mousetype)
{
      union REGS workregs;
      struct SREGS sregs;

      /* check the vector     */

      segread (&sregs);
      workregs.h.ah = 0x35;     /* DOS get vector */
      workregs.h.al = 0x33;     /* mouse vector   */
      intdosx(&workregs, &workregs, &sregs);

      /* ES:BX now contains the pointer to the interrupt handler */

      if (sregs.es == 0 && workregs.x.bx == 0)
            return mouse_present = 0;

      workregs.x.ax = 0;
      MOUSE(workregs);
      *mousetype = workregs.x.bx;
      mouse_present = workregs.x.ax;
      return(mouse_present);
}

/*
** Makes the mouse cursor visible.
*/

void ms_show_cursor(void)
{
      union REGS workregs;

      workregs.x.ax = 1;
      MOUSE(workregs);
}

/*
** Hides the mouse cursor.  Should be called before changing any portion of
** the screen under the mouse cursor.
*/

void ms_hide_cursor(void)
{
      union REGS workregs;

      workregs.x.ax = 2;
      MOUSE(workregs);
}

/*
** Obtains information about the mouse position and button status.
** Places the current horizontal and vertical positions in *horizpos and
** *vertpos, respectively.  Returns the mouse button status, which is
** mapped at the bit level as follows:
**    Bit 0 - left button    \
**    Bit 1 - right button    >-- 0 = button up, 1 = button down
**    Bit 2 - middle button  /
*/

int ms_get_mouse_pos(int *horizpos, int *vertpos) /* Returns button status */
{
      union REGS workregs;

      workregs.x.ax = 3;
      MOUSE(workregs);
      *horizpos = workregs.x.cx;
      *vertpos  = workregs.x.dx;
      return(workregs.x.bx);
}

/*
```

```
101 | ** Moves the mouse cursor to a new position.
102 | */
103 |
104 | void ms_set_mouse_pos(int horizpos, int vertpos)
105 | {
106 |       union REGS workregs;
107 |
108 |       workregs.x.ax = 4;
109 |       workregs.x.cx = horizpos;
110 |       workregs.x.dx = vertpos;
111 |       MOUSE(workregs);
112 | }
113 |
114 | /*
115 | ** Obtains information about the last time the specified button
116 | ** (0 = left, 1 = right, 2 = middle) was pressed.  Returns the current
117 | ** button status (same format as return from ms_get_mouse_pos() above).
118 | */
119 |
120 | int ms_button_press_status(int  button,
121 |                            int *press_count,
122 |                            int *column,
123 |                            int *row)
124 | {
125 |       union REGS workregs;
126 |
127 |       workregs.x.ax = 5;
128 |       workregs.x.bx = button;
129 |       MOUSE(workregs);
130 |       *press_count = workregs.x.bx;
131 |       *column = workregs.x.cx;
132 |       *row = workregs.x.dx;
133 |       return(workregs.x.ax);
134 | }
135 |
136 | /*
137 | ** Similar to above but obtains information about the last release of the
138 | ** specified button.
139 | */
140 |
141 | int ms_button_release_status(int  button,
142 |                              int *release_count,
143 |                              int *column,
144 |                              int *row)
145 | {
146 |       union REGS workregs;
147 |
148 |       workregs.x.ax = 6;
149 |       workregs.x.bx = button;
150 |       MOUSE(workregs);
151 |       *release_count = workregs.x.bx;
152 |       *column = workregs.x.cx;
153 |       *row = workregs.x.dx;
154 |       return(workregs.x.ax);
155 | }
156 |
157 | /*
158 | ** Forces the mouse cursor to remain within the range specified.
159 | */
160 |
161 | void ms_restrict_horiz(int min, int max)
162 | {
163 |       union REGS workregs;
164 |
165 |       workregs.x.ax = 7;
166 |       workregs.x.cx = min;
167 |       workregs.x.dx = max;
168 |       MOUSE(workregs);
169 | }
170 |
171 | /*
172 | ** Forces the mouse cursor to remain within the range specified.
173 | */
174 |
175 | void ms_restrict_vert(int min, int max)
176 | {
177 |       union REGS workregs;
178 |
179 |       workregs.x.ax = 8;
180 |       workregs.x.cx = min;
181 |       workregs.x.dx = max;
182 |       MOUSE(workregs);
183 | }
184 |
185 | void ms_define_window(int left, int top, int right, int bottom)
186 | {
187 |       ms_restrict_horiz(left,right);
188 |       ms_restrict_vert(top,bottom);
189 | }
190 |
191 | /*
192 | ** Allows the user to set the graphics cursor to a new shape.  Check your
193 | ** mouse reference manual for full information about the use of this function.
194 | */
195 |
196 | void ms_set_graphics_cursor(int      horiz_hotspot,
197 |                             int      vert_hotspot,
198 |                             unsigned seg_shape_tables,
199 |                             unsigned offset_shape_tables)
200 | {
```

```
201 |       union REGS workregs;
202 |       struct SREGS segregs;
203 |
204 |       workregs.x.ax = 9;
205 |       workregs.x.bx = horiz_hotspot;
206 |       workregs.x.cx = vert_hotspot;
207 |       workregs.x.dx = offset_shape_tables;
208 |       segregs.es  = seg_shape_tables;
209 |       MOUSEX(workregs, segregs);
210 | }
211 |
212 | /*
213 | ** Selects either the software or hardware cursor and sets the start and stop
214 | ** scan lines (for the hardware cursor) or the screen and cursor masks (for
215 | ** the software cursor).  Consult your mouse reference for more information.
216 | */
217 |
218 | void ms_set_text_cursor(int type, int screen_mask, int cursor_mask)
219 | {
220 |       union REGS workregs;
221 |
222 |       workregs.x.ax = 10;
223 |       workregs.x.bx = type;
224 |       workregs.x.cx = screen_mask;
225 |       workregs.x.dx = cursor_mask;
226 |       MOUSE(workregs);
227 | }
228 |
229 | /*
230 | ** Obtains the horizontal and vertical raw motion counts since the last
231 | ** request.
232 | */
233 |
234 | void ms_read_motion_counters(int *horiz, int *vert)
235 | {
236 |       union REGS workregs;
237 |
238 |       workregs.x.ax = 11;
239 |       MOUSE(workregs);
240 |       *horiz = workregs.x.cx;
241 |       *vert  = workregs.x.dx;
242 | }
243 |
244 | /*
245 | ** Sets up a subroutine to be called when a given event occurs.
246 | ** NOTE: Use with extreme care.  The function whose address is provided MUST
247 | **    terminate with a far return (i.e. must be compiled using large model).
248 | **    Also, no DOS or BIOS services may be used, as the user-defined function
249 | **    is (in effect) an extension to an interrupt service routine.
250 | */
251 |
252 | void ms_set_event_subroutine(int      mask,
253 |                              unsigned seg_routine,
254 |                              unsigned offset_routine)
255 | {
256 |       union REGS workregs;
257 |       struct SREGS segregs;
258 |
259 |       workregs.x.ax = 12;
260 |       workregs.x.cx = mask;
261 |       workregs.x.dx = offset_routine;
262 |       segregs.es  = seg_routine;
263 |       MOUSEX(workregs, segregs);
264 | }
265 |
266 | /*
267 | ** Turns light pen emulation mode on.
268 | */
269 |
270 | void ms_light_pen_on(void)
271 | {
272 |       union REGS workregs;
273 |
274 |       workregs.x.ax = 13;
275 |       MOUSE(workregs);
276 | }
277 |
278 | /*
279 | ** turns light pen emulation mode off.
280 | */
281 |
282 | void ms_light_pen_off(void)
283 | {
284 |       union REGS workregs;
285 |
286 |       workregs.x.ax = 14;
287 |       MOUSE(workregs);
288 | }
289 |
290 | /*
291 | ** Sets the sensitivity of the mouse.  Defaults are 8 and 16 for horizontal
292 | ** and vertical sensitivity (respectively).
293 | */
294 |
295 | void ms_set_sensitivity(int horiz, int vert)
296 | {
297 |       union REGS workregs;
298 |
299 |       workregs.x.ax = 15;
300 |       workregs.x.cx = horiz;
```

```
301 |       workregs.x.dx = vert;
302 |       MOUSE(workregs);
303 | }
304 |
305 | /*
306 | ** Sets up a region of the screen inside of which the mouse cursor will
307 | ** automatically be 'hidden'.
308 | */
309 |
310 | void ms_protect_area(int left, int top, int right, int bottom)
311 | {
312 |       union REGS workregs;
313 |
314 |       workregs.x.ax = 16;
315 |       workregs.x.cx = left;
316 |       workregs.x.dx = top;
317 |       workregs.x.si = right;
318 |       workregs.x.di = bottom;
319 |       MOUSE(workregs);
320 | }
321 |
322 | /*
323 | * Similar to ms_set_graphics_cursor() but allows a larger cursor.  Consult
324 | ** your mouse documentation for information on how to use this function.
325 | */
326 |
327 | int ms_set_large_graphics_cursor(int      width,
328 |                                  int      height,
329 |                                  int      horiz_hotspot,
330 |                                  int      vert_hotspot,
331 |                                  unsigned seg_shape_tables,
332 |                                  unsigned offset_shape_tables)
333 | {
334 |       union REGS workregs;
335 |       struct SREGS segregs;
336 |
337 |       workregs.x.ax = 18;
338 |       workregs.x.bx = (width << 8) + horiz_hotspot;
339 |       workregs.x.cx = (height << 8) + vert_hotspot;
340 |       workregs.x.dx = offset_shape_tables;
341 |       segregs.es  = seg_shape_tables;
342 |       MOUSEX(workregs, segregs);
343 |       if(workregs.x.ax == (unsigned)-1)
344 |             return(workregs.x.ax); /* Return -1 if function 18 supported */
345 |       else  return(0);             /* else return 0                      */
346 | }
347 |
348 | /*
349 | ** Sets the threshold value for doubling cursor motion.  Default value is 64.
350 | */
351 |
352 | void ms_set_doublespeed_threshold(int speed)
353 | {
354 |       union REGS workregs;
355 |
356 |       workregs.x.ax = 19;
357 |       workregs.x.dx = speed;
358 |       MOUSE(workregs);
359 | }
```

TEXT STATISTICS

9281 characters
359 lines

LEXICAL STATISTICS

29 comments [std-C]
0 comments [C++]
5 preprocessor instructions
0 constants [character]
1 constants [string]
30 constants [numeric]

SYMBOL TABLE

```
      209   220   222   223   224   225   226   236   238   239   240   241
      256   259   260   261   263   272   274   275   284   286   287   297
      299   300   301   302   312   314   315   316   317   318   319   334
      337   338   339   340   342   343   344   354   356   357   358
x      44    47    49    50    62    75    93    95    96    97   108   109
      110   127   128   130   131   132   133   148   149   151   152   153
      154   165   166   167   179   180   181   204   205   206   207   222
      223   224   225   238   240   241   259   260   261   274   286   299
      300   301   314   315   316   317   318   337   338   339   340   343
      344   356   357
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*      module:         mouse.h
  4 |  *      programmer:     Ray L. McVay
  5 |  *      started:        26oct86
  6 |  *      updated:        26oct86
  7 |  *
  8 |  *      Some handy mouse interface functions.
  9 |  */
 10 |
 11 | #ifndef MOUSE__H
 12 | #define MOUSE__H
 13 |
 14 | #include "extkword.h"
 15 |
 16 | #define MSMOUSE 0x33
 17 |
 18 | extern int mouse_present;
 19 |
 20 | int  ms_reset(int *);
 21 | void ms_show_cursor(void);
 22 | void ms_hide_cursor(void);
 23 | int  ms_get_mouse_pos(int *, int *);
 24 | void ms_set_mouse_pos(int, int);
 25 | int  ms_button_press_status(int, int *, int *, int *);
 26 | int  ms_button_release_status(int, int *, int *, int *);
 27 | void ms_restrict_horiz(int, int);
 28 | void ms_restrict_horiz(int, int);
 29 | void ms_define_window(int, int, int, int);
 30 | void ms_set_graphics_cursor(int, int, unsigned, unsigned);
 31 | void ms_set_text_cursor(int, int, int);
 32 | void ms_read_motion_counters(int *, int *);
 33 | void ms_set_event_subroutine(int, unsigned, unsigned);
 34 | void ms_light_pen_on(void);
 35 | void ms_light_pen_off(void);
 36 | void ms_set_sensitivity(int, int);
 37 | void ms_protect_area(int, int, int, int);
 38 | int  ms_set_large_graphics_cursor(int, int, int, int, unsigned, unsigned);
 39 | void ms_set_doublespeed_threshold(int);
 40 |
 41 | #endif /* MOUSE__H */
```

TEXT STATISTICS

```
  1216 characters
    41 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    MOUSE__H        11      12
    MSMOUSE         16
    mouse_present           18
    ms_button_press_status          25
    ms_button_release_status        26
    ms_define_window        29
    ms_get_mouse_pos        23
    ms_hide_cursor          22
    ms_light_pen_off        35
    ms_light_pen_on         34
    ms_protect_area         37
    ms_read_motion_counters         32
    ms_reset        20
    ms_restrict_horiz       27      28
    ms_set_doublespeed_threshold            39
    ms_set_event_subroutine         33
    ms_set_graphics_cursor          30
    ms_set_large_graphics_cursor            38
    ms_set_mouse_pos        24
    ms_set_sensitivity              36
    ms_set_text_cursor              31
    ms_show_cursor          21
```

```
/* +++Date last modified: 05-Jul-1997 */

/***  MSBIN conversion routines    ***/
/***  public domain by Jeffery Foy ***/

#include "snipmath.h"

union Converter {
      unsigned char uc[10];
      unsigned int  ui[5];
      unsigned long ul[2];
      float          f[2];
      double         d[1];
};

/* MSBINToIEEE - Converts an MSBIN floating point number */
/*               to IEEE floating point format           */
/*                                                       */
/*  Input: f - floating point number in MSBIN format     */
/* Output: Same number in IEEE format                    */

float MSBINToIEEE(float f)
{
      union Converter t;
      int sign, exp;       /* sign and exponent */

      t.f[0] = f;

      /* extract the sign & move exponent bias from 0x81 to 0x7f */

      sign = t.uc[2] / 0x80;
      exp  = (t.uc[3] - 0x81 + 0x7f) & 0xff;

      /* reassemble them in IEEE 4 byte real number format */

      t.ui[1] = (t.ui[1] & 0x7f) | (exp << 7) | (sign << 15);
      return t.f[0];
} /* End of MSBINToIEEE */


/* IEEEToMSBIN - Converts an IEEE floating point number  */
/*               to MSBIN floating point format          */
/*                                                       */
/*  Input: f - floating point number in IEEE format      */
/* Output: Same number in MSBIN format                   */

float IEEEToMSBIN(float f)
{
      union Converter t;
      int sign, exp;       /* sign and exponent */

      t.f[0] = f;

      /* extract sign & change exponent bias from 0x7f to 0x81 */

      sign = t.uc[3] / 0x80;
      exp  = ((t.ui[1] >> 7) - 0x7f + 0x81) & 0xff;

      /* reassemble them in MSBIN format */

      t.ui[1] = (t.ui[1] & 0x7f) | (sign << 7) | (exp << 8);
      return t.f[0];
} /* End of IEEEToMSBIN */
```

TEXT STATISTICS

```
  1841 characters
    63 lines
```

LEXICAL STATISTICS

```
    21 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
    32 constants [numeric]
```

SYMBOL TABLE

```
    Converter        8      24      49
    IEEEToMSBIN             47
    MSBINToIEEE             22
    d       13
    exp             25      32      36      50      57      61
    f       12      22      27+     37      47      52+     62
    sign            25      31      36      50      56      61
    t       24      27      31      32      36+     37      49      52      56      57      61+     62
    uc               9      31      32      56
    ui              10      36+     57      61+
    ul              11
```

```
/* +++Date last modified: 05-Jul-1997 */

/* MTERM.C - Minimal example PC terminal program.
   Released to public domain by author, David Harmon, June 1992.
   Intended for use with a FOSSIL driver, but will run with BIOS alone.
   I expect you'll want to add something for practical purposes. ;-)
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>            /* kbhit(), getch(), putch(), etc. */
#include <dos.h>              /* int86(), etc. */

#ifdef __ZTC__
 #define cputs(s) fputs((s),stderr)
#endif

int port = 0;                 /* 0 = COM1:, 1 = COM2: etc. */
int local_echo = 0;
int cr_add_lf = 0;
int exiting = 0;

int init_comm(int flags)
{
      union REGS regs;

      regs.h.ah = 0x04;         /* initialize driver (port) */
      regs.x.bx = 0x4f50;
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);

      regs.h.ah = 0x00;         /* set baud rate * port attrs */
      regs.h.al = (unsigned char)flags;
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);
      return regs.h.ah;
}

void send_char(char ch)
{
      union REGS regs;

      regs.h.ah = 0x01;         /* Send char (wait until ready)*/
      regs.h.al = ch;
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);
}

int   input_ready(void)
{
      union REGS regs;

      regs.h.ah = 0x03;         /* Get port status */
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);
      return ((regs.h.ah & 0x01) != 0);   /* input ready */
}

int   get_char(void)
{
      union REGS regs;

      regs.h.ah = 0x02;         /* receive char (wait if necessary)*/
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);
      return regs.h.al;
}

void deinit_comm(void)
{
      union REGS regs;

      regs.h.ah = 0x05;         /* deinitialize port (pseudo close) */
      regs.h.al = 0x00;         /* (lower DTR) */
      regs.x.dx = port;
      int86( 0x14, &regs, &regs);
}

main()
{
      int ch;

      init_comm(0xE3);          /* hard coded 0xB3 = 2400,N,8,1 */
      cputs("MTERM ready!   Press F1 to exit.\r\n");
      while (!exiting)
      {
            if (kbhit())        /* key was hit */
            {
                  ch = getch();     /* Regular ASCII keys are returned as the
                                       ASCII code; function keys, arrows, etc.
                                       as zero followed by a special code
                                       (on next getch.)  */
                  if (ch != 0)
                  {
                        send_char((char)ch);    /* to com port */
                        if (local_echo)
                        {
                              putch(ch);        /* to screen */

                              /* add LF to CR? */
```

```
101 |
102 |                                 if (cr_add_lf && ch == '\r')
103 |                                       putch('\n');
104 |                           }
105 |                     }
106 |                     else
107 |                     {
108 |                           ch = getch();       /* get the special key code */
109 |                           switch (ch)
110 |                           {
111 |                           case 0x3B: /* F1 */
112 |                                 exiting = 1;              /* quit now */
113 |                                 break;
114 |
115 |                           case 0x3C: /* F2 */
116 |                                 local_echo = !local_echo;  /* toggle echo */
117 |                                 break;
118 |
119 |                           case 0x3D: /* F3 */
120 |                                 cr_add_lf = !cr_add_lf;    /* toggle LF */
121 |                                 break;
122 |                           }
123 |                     }
124 |               } /* end if kbhit */
125 |
126 |               if (input_ready())       /* com port */
127 |               {
128 |                     ch = get_char();
129 |                     putch(ch);
130 |                     if (cr_add_lf && ch == '\r')  /* add LF to CR? */
131 |                           putch('\n');
132 |               }
133 |         } /* end while not exiting */
134 |         deinit_comm();
135 |         return 0;
136 | }
```

TEXT STATISTICS

```
  4056 characters
   136 lines
```

LEXICAL STATISTICS

```
    30 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     4 constants [character]
     1 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    REGS            25      41      51      61      71
    __ZTC__         14
    ah              27      32      36      43      53      56      63      73
    al              33      44      66      74
    bx              28
    ch              39      44      81      89      93      95      98     102     108     109     128
           129     130
    conio           11
    cputs           15      84
    cr_add_lf       20     102     120+    130
    deinit_comm              69     134
    dos             12
    dx              29      34      45      54      64      75
    exiting         21      85     112
    flags           23      33
    fputs           15
    get_char        59     128
    getch           89     108
    h        9      10      11      12      27      32      33      36      43      44      53      56
            63      66      73      74
    init_comm       23      83
    input_ready              49     126
    int86           30      35      46      55      65      76
    kbhit           87
    local_echo               19      96     116+
    main            79
    port            18      29      34      45      54      64      75
    putch           98     103     129     131
    regs            25      27      28      29      30+     32      33      34      35+     36      41
           43       44      45      46+     51      53      54      55+     56      61      63      64
           65+      66      71      73      74      75      76+
    s       15+
    send_char       39      95
    stderr          15
    stdio            9
    stdlib          10
    x       28      29      34      45      54      64      75
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** mv.c -- move or rename files or directories
** updated for multiple files, 5 jul 92, rlm
** placed in the public domain via C_ECHO by the author, Ray McVay
**
** modified by Bob Stout, 28 Mar 93
** modified by Bob Stout,  4 Jun 93
**
** uses file_copy from SNIPPETS file WB_FCOPY.C
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include "dirport.h"
#include "snipfile.h"
#include "dosfiles.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

/*
**  Tell 'em they messed up
*/

void help(char *s)
{
      puts("usage: mv <oldname [...]> <newname|newdir>");
      printf("error: %s\n", s);
}

/*
**  Simple directory test
*/

isdir(char *path)
{
      DOSFileData f;

      /* "Raw" drive specs are always directories     */

      if (':' == path[1] && '\0' == path[2])
            return 1;

      return (Success_ == FIND_FIRST(path, _A_SUBDIR, &f) &&
            (ff_attr(&f) & _A_SUBDIR));
}

/*
**  Use rename or copy and delete
*/

int mv(char *src, char *dest)
{
      int errcount = 0;
      char buf[FILENAME_MAX];
      const char *generr = "ERROR: mv - couldn't %s %s %s\n";

      if (':' == dest[1] && *dest != *getcwd(buf, FILENAME_MAX))
      {
            if (file_copy(src, dest))
            {
                  printf(generr, "move", src, dest);
                  ++errcount;
            }
            else if (remove(src))
            {
                  printf(generr, "delete", src, "");
                  ++errcount;
            }
      }
      else
      {
            if (rename(src, dest))
            {
                  printf(generr, "rename", src, dest);
                  ++errcount;
            }
      }
      return errcount;
}

/*
**  Enter here
*/

int main(int argc, char **argv)
{
      int src, errcount = 0;
      char target[FILENAME_MAX];

      puts("mv 1.3 (4 jun 93) - Ray L. McVay/Bob Stout");
      if (argc < 3)
            help("Not enough parameters");

```

```
101 |       /*
102 |       **  Handle cases where target is a directory
103 |       */
104 |
105 |       else if (isdir(argv[argc -1]))
106 |       {
107 |             for (src = 1; src < argc - 1; src++)
108 |             {
109 |                   char termch;
110 |
111 |                   strcpy(target, argv[argc - 1]);
112 |                   termch = target[strlen(target) - 1];
113 |                   if ('\\' != termch && ':' != termch)
114 |                         strcat(target, "\\");
115 |
116 |                   if (strrchr(argv[src], '\\'))
117 |                         strcat(target, strrchr(argv[src], '\\') + 1);
118 |                   else if (argv[src][1] == ':')
119 |                         strcat(target, argv[src] + 2);
120 |                   else  strcat(target, argv[src]);
121 |
122 |                   errcount += mv(argv[src], target);
123 |             }
124 |       }
125 |
126 |       /*
127 |       **  Nothing left except 2 explicit file names
128 |       */
129 |
130 |       else if (argc == 3)
131 |             errcount += mv(argv[1], argv[2]);
132 |
133 |       return errcount;
134 | }
```

TEXT STATISTICS

```
  3108 characters
   134 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     8 constants [character]
    14 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             43
    FILENAME_MAX            61      64      95
    FIND_FIRST              50
    MSDOS           21
    Success_        50
    _A_SUBDIR       50      51
    __MSDOS__       21
    argc            92      98     105     107     111     130
    argv            92     105     111     116     117     118     119     120     122     131+
    buf             61      64
    defined         21+
    dest            58      64+     66      68      79      81
    dos             17
    errcount        60      69      74      82      85      94     122     131     133
    f       43      50      51
    ff_attr         51
    file_copy       66
    generr          62      68      73      81
    getcwd          64
    h       14      15      16      17      24
    help            31      99
    isdir           41     105
    main            92
    mv              58     122     131
    path            41      47+     50
    printf          34      68      73      81
    puts            33      97
    remove          71
    rename          79
    s       31      34
    src             58      66      68      71      73      79      81      94     107+    116     117
           118     119     120     122
    stdio           14
    stdlib          15
    strcat         114     117     119     120
    strcpy         111
    string          16
    strlen         112
    strrchr        116     117
    target          95     111     112+    114     117     119     120     122
    termch         109     112     113+
    unistd          24
```

```
/* +++Date last modified: 05-Jul-1997 */

// Myio.h
// Specialised I/O class to demonstrate use of C++ iostream
// facilities in a customised environment
// Written by David L Nugent, June 1993.
//

# if !defined(_Myio_h)
# define _Myio_h 1

    // Forward declare classes

class Myio;
class Mystreambuf;
class Mystreambase;
class Mystream;

    // Forward declare iostream classes

class iostream;

    //
    // class Myio
    // This is a simplistic class which simply fields
    // input and output to a simulated stream device.
    //
    // In fact, it doesn't really do much at all other
    // than read input from and send output to a
    // circular queue, as though talking via a loopback
    // pipe to itself.
    //


class Myio
{
    friend class Mystreambuf;

  public:

    Myio (int sz =2048);                    // sz = buffer size to allocate
    virtual ~Myio (void);

    iostream & stream (void);               // Return (or create) stream

    int readok (void) const;                // Underflow check
    int writeok (void) const;               // Overflow check
    int gcount (void) const;                // Get # of chrs last read
    int pcount (void) const;                // Get # of chrs last written
    int count (void) const;                 // Get # of chrs in buffer
    int size (void) const;                  // Get size of buffer
    int dump (void) const;                  // Debugging - dumps buffer

    int write (char const * buf, int len);  // Put data into 'pipe'
    int read (char * buf, int max);         // Read data from our 'pipe'

  private:

    enum
    {
        overflow    = 0x0001,   // Last write only partial
        underflow   = 0x0002    // Last read only partial
    };

    unsigned stat;              // Last read/write status
    int _pcount;                // Last write count
    int _gcount;                // Last read count
    int bufsize;                // Size of our buffer
    int bufchars;               // Chrs in buffer now
    int bufidx;                 // Index into buffer (next put)
    char * bufaddr;             // Pointer to buffer
    Mystream * mystream;        // Stream associated with this object

};

inline int
Myio::readok (void) const
    {   return ((stat & Myio::underflow) == 0); }

inline int
Myio::writeok (void) const
    {   return ((stat & Myio::overflow) == 0);  }

inline int
Myio::gcount (void) const
    {   return _gcount;     }

inline int
Myio::pcount (void) const
    {   return _pcount;     }

inline int
Myio::count (void) const
    {   return bufchars;    }

inline int
Myio::size (void) const
    {   return bufsize;     }

# endif     // _Myio_h
```

TEXT STATISTICS

```
  2768 characters
   100 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
    39 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    Myio            14      35      41      42      77      78      81      82      85      89      93
           97
    Mystream        17      72
    Mystreambase            16
    Mystreambuf             15      37
    _Myio_h          9      10
    _gcount         67      86
    _pcount         66      90
    buf             54      55
    bufaddr         71
    bufchars        69      94
    bufidx          70
    bufsize         68      98
    count           50      93
    define          10
    defined          9
    dump            52
    endif          100
    gcount          48      85
    iostream        21      44
    len             54
    max             55
    mystream        72
    overflow        61      82
    pcount          49      89
    read            55
    readok          46      77
    size            51      97
    stat            65      78      82
    stream          44
    sz              41
    underflow       62      78
    write           54
    writeok         47      81
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | // myLine.h
   4 | //
   5 | // Donated to the public domain; no restrictions on reuse or abuse apply.
   6 | // by David Nugent, 7th June, 1993.
   7 | // Simple line input class for istream to demonstrate input of a complete
   8 | // line rather than whitespace separated tokens (the default for operator<<
   9 | // for char* and other built-in types).
  10 | // Works by overloading operator>> for a customised class - this functionality
  11 | // is easily incorporated into your favourite String class
  12 | //
  13 |
  14 | # if !defined(_myLine_h)
  15 | # define _myLine_h 1
  16 |
  17 | # define AUTO_GROW 1            // Allow autogrowth of buffer to fit
  18 | # define ALLOC_LEN 80           // Standard length & growth increment
  19 |
  20 |     // Class declaration
  21 |
  22 | class myLine
  23 | {
  24 |
  25 |   public:
  26 |
  27 |     myLine (short buflen =ALLOC_LEN);
  28 |     myLine (char * usebuf, short buflen =ALLOC_LEN);
  29 |     ~myLine (void);
  30 |                                       // Get buffer address
  31 |     char const * buf (void) const { return mybuf; }
  32 |                                       // Conversion operators
  33 |     char const * operator() (void) const { return mybuf; } // Explicit cast
  34 |     operator char const * (void) const { return mybuf; }   // Implicit cast
  35 |                                       // istream operator>>
  36 |     friend istream & operator>> (istream &, myLine &);
  37 |
  38 |   private:
  39 |
  40 |     short len, xalloc;
  41 |     char * mybuf;
  42 |
  43 | };
  44 |
  45 | # endif     // _myLine_h
```

TEXT STATISTICS

```
  1423 characters
    45 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
    19 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    ALLOC_LEN       18      27      28
    AUTO_GROW       17
    _myLine_h       14      15
    buf             31
    buflen          27      28
    define          15      17      18
    defined         14
    endif           45
    istream         36+
    len             40
    myLine          22      27      28      29      36
    mybuf           31      33      34      41
    usebuf          28
    xalloc          40
```

```
/* +++Date last modified: 05-Jul-1997 */

// Mystream.h
// iostream interface for class Myio
// Defines the following classes:
//  Mystreambuf     derived from streambuf - buffer management & I/O interface
//  Mystreambase    base class used for initialisation & object reference
//  Myiostream      customised iostream, derived from iostream/Mystreambase
//
// Written by David L Nugent, June 1993
//

# if !defined(_Mystream_h)
# define _Mystream_h 1
# include <iostream.h>
# include "Myio.h"

    //
    // Mystreambuf
    // This is the class which does all the actual I/O
    // handling and (optional) buffer management
    //

class Mystreambuf : public streambuf
{

  public:

    Mystreambuf (Myio * mPtr);

  protected:

    virtual int overflow (int = EOF);
    virtual int underflow ();
    virtual int sync ();

  private:

    Myio * mptr;    // Points to the Myio instance to
                    // which this stream is attached
    char _back[2];  // Holder for putback

};


class Mystreambase : public virtual ios
{

  public:

    Mystreambase (Myio * mPtr);
    Mystreambuf * rdbuf (void);

  protected:

    Mystreambuf mystreambuf;

};

inline
Mystreambase::Mystreambase (Myio * mPtr)
    : mystreambuf (mPtr)
{}

inline Mystreambuf *
Mystreambase::rdbuf (void)
    {   return &mystreambuf;    }


class Mystream : public Mystreambase, public iostream
{

  public:

    Mystream (Myio * mPtr);
    ~Mystream (void);
};

    //
    // class Mystream constructor
    // This uses Mystreambase to set up the Mystreambuf
    // which can then be used to initialise iostream.
    //

inline
Mystream::Mystream (Myio * m)
    : Mystreambase (m), iostream (rdbuf())
{}

inline
Mystream::~Mystream (void)
    {}

# endif     // _Mystream_h
```

TEXT STATISTICS

```
  1848 characters
    94 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
    23 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     1 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    EOF             33
    Myio            29      39      51      61      75      86
    Mystream        70      75      76      86+     91+
    Mystreambase            46      51      61+     66      70      87
    Mystreambuf             24      29      52      56      65
    _Mystream_h             13      14
    _back           41
    define          14
    defined         13
    endif           94
    h       15
    include         15      16
    ios             46
    iostream        15      70      87
    m       86      87
    mPtr            29      51      61      62      75
    mptr            39
    mystreambuf             56      62      67
    overflow        33
    rdbuf           52      66      87
    streambuf       24
    sync            35
    underflow       34
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  NLCNVRT.C - A utility to convert newlines in text files
**
**  public domain by Bob Stout
**  Mac support by Norman Dodge and Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errors.h"
#include "sniptype.h"

FILE *infile;
FILE *outfile;

static void usage(void);
static void dos2unix(void);
static void dos2mac(void);
static void unix2dos(void);
static void unix2mac(void);
static void mac2dos(void);
static void mac2unix(void);
static void Cout(int ch, FILE *outfile);
static void Sout(char *str, FILE *outfile);

main(int argc, char *argv[])
{
      char outname[L_tmpnam] = "";

      if (argc < 3)
            usage();
      if (NULL == strchr("-/", argv[1][0]))
            usage();

      infile = cant(argv[2], "rb");

      if (3 == argc || NULL == (outfile = fopen(argv[3], "wb")))
      {
            tmpnam(outname);
            outfile = cant(outname, "wb");
      }

      switch (argv[1][1])
      {
      case 'd':
            unix2dos();
            break;

      case 'D':
            mac2dos();
            break;

      case 'u':
            dos2unix();
            break;

      case 'U':
            mac2unix();
            break;

      case 'm':
            dos2mac();
            break;

      case 'M':
            unix2mac();
            break;

      default:
            usage();
      }
      fclose(infile);
      fclose(outfile);
      if (*outname)
      {
            if (Success_ != remove(argv[2]))
            {
                  remove(outname);
                  ErrExit("Can't remove %s", argv[2]);
            }
            rename(outname, argv[2]);
      }
      return EXIT_SUCCESS;
}

void usage(void)
{
      puts("Usage: NLCNVRT -[d | u | m | D | U | M] infile [outfile]");
      puts("Switches: -d  Convert to DOS  from Unix - converts LF   => CRLF");
      puts("        : -u  Convert to Unix from DOS  - converts CRLF => LF");
      puts("        : -m  Convert to Mac  from DOS  - converts CRLF => CR");
      puts("        : -D  Convert to DOS  from Mac  - converts CR   => CRLF");
      puts("        : -U  Convert to Unix from Mac  - converts CR   => LF");
      puts("        : -M  Convert to Mac  from Unix - converts LF   => CR");
      exit(EXIT_FAILURE);
}
```

```
101 | void unix2dos(void)
102 | {
103 |       int ch, lastch = 0;
104 |
105 |       while (EOF != (ch = fgetc(infile)))
106 |       {
107 |             if ('\n' == ch && '\r' != lastch)
108 |                   fputc('\r', outfile);
109 |             fputc(lastch = ch, outfile);
110 |       }
111 |       if ('\n' != lastch)
112 |             fputs("\r\n", outfile);
113 | }
114 |
115 | void dos2unix(void)
116 | {
117 |       int ch, lastch = 0;
118 |
119 |       while (EOF != (ch = fgetc(infile)) && '\x1a' != ch)
120 |       {
121 |             if ('\r' != ch)
122 |                   fputc(lastch = ch, outfile);
123 |       }
124 |       if ('\n' != lastch)
125 |             fputc('\n', outfile);
126 | }
127 |
128 | void mac2dos(void)
129 | {
130 |       int ch, lastch = 0;
131 |
132 |       while (EOF != (ch = fgetc(infile)))
133 |       {
134 |             fputc(lastch = ch, outfile);
135 |             if ('\r' == ch)
136 |                   fputc('\n', outfile);
137 |       }
138 |       if ('\r' != lastch)
139 |             fputs("\r\n", outfile);
140 | }
141 |
142 | static void dos2mac(void)
143 | {
144 |       int ch, lastch = 0;
145 |
146 |       while (EOF != (ch = fgetc(infile)) && '\x1a' != ch)
147 |       {
148 |             if ('\r' != ch)
149 |             {
150 |                   if ('\n' == ch)
151 |                         ch = '\r';
152 |                   fputc(lastch = ch, outfile);
153 |             }
154 |       }
155 |       if ('\r' != lastch)
156 |             fputc('\r', outfile);
157 | }
158 |
159 | static void unix2mac(void)
160 | {
161 |       int ch, lastch = 0;
162 |
163 |       while (EOF != (ch = fgetc(infile)))
164 |       {
165 |             if ('\n' == ch)
166 |                   ch = '\r';
167 |             fputc(lastch = ch, outfile);
168 |       }
169 |       if ('\r' != lastch)
170 |             fputc('\r', outfile);
171 | }
172 |
173 | static void mac2unix(void)
174 | {
175 |       int ch, lastch = 0;
176 |
177 |       while (EOF != (ch = fgetc(infile)))
178 |       {
179 |             if ('\r' == ch)
180 |                   ch = '\n';
181 |             fputc(lastch = ch, outfile);
182 |       }
183 |       if ('\n' != lastch)
184 |             fputc('\n', outfile);
185 | }
186 |
187 | void Cout(int ch, FILE *outfile)
188 | {
189 |       if (EOF == fputc(ch, outfile))
190 |             exit(EXIT_FAILURE);
191 | }
192 |
193 | void Sout(char *str, FILE *outfile)
194 | {
195 |       if (EOF == fputs(str, outfile))
196 |             exit(EXIT_FAILURE);
197 | }
```

TEXT STATISTICS

```
  4465 characters
   197 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
    31 constants [character]
    17 constants [string]
    17 constants [numeric]
```

SYMBOL TABLE

```
    Cout            26     187
    EOF            105     119     132     146     163     177     189     195
    EXIT_FAILURE            98     190     196
    EXIT_SUCCESS            86
    ErrExit         82
    FILE            16      17      26      27     187     193
    L_tmpnam        31
    NULL            35      40
    Sout            27     193
    Success_        79
    argc            29      33      40
    argv            29      35      38      40      46      79      82      84
    cant            38      43
    ch              26     103     105     107     109     117     119+    121     122     130     132
           134     135     144     146+    148     150     151     152     161     163     165     166
           167     175     177     179     180     181     187     189
    dos2mac         21      65     142
    dos2unix        20      57     115
    exit            98     190     196
    fclose          75      76
    fgetc          105     119     132     146     163     177
    fopen           40
    fputc          108     109     122     125     134     136     152     156     167     170     181
           184     189
    fputs          112     139     195
    h        10     11      12
    infile          16      38      75     105     119     132     146     163     177
    lastch         103     107     109     111     117     122     124     130     134     138     144
           152     155     161     167     169     175     181     183
    mac2dos         24      53     128
    mac2unix        25      61     173
    main            29
    outfile         17      26      27      40      43      76     108     109     112     122     125
           134     136     139     152     156     167     170     181     184     187     189     193
           195
    outname         31      42      43      77      81      84
    puts            91      92      93      94      95      96      97
    remove          79      81
    rename          84
    stdio           10
    stdlib          11
    str             27     193     195
    strchr          35
    string          12
    tmpnam          42
    unix2dos        22      49     101
    unix2mac        23      69     159
    usage           19      34      36      73      89
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ----------------------------------------------------------------------
   Module:     ntstream.h
   Subject:    Share-Aware File Streams
   Author:     Heinz Ozwirk
   Started:    30.05.1993 15:22:54
   Modified:   31.05.1993 10:28:24
   ----------------------------------------------------------------------
   Description: public domain from the FidoNet C++ echo
   ----------------------------------------------------------------------
   History:    (insert new entries at top of list)
   dd.mm.yyyy/ho description
   ---------------------------------------------------------------------- */

#if !defined __NETSTREAM_H
#define __NETSTREAM_H

/* --- Includes ------------------------------------------------------- */
#include <fstream.h>
#include <share.h>

/* --- Defines -------------------------------------------------------- */
/* --- Constants ------------------------------------------------------ */
/* --- Types ---------------------------------------------------------- */

class nfstream: public fstream
   {
   public:
      enum
         {
         sh_compat = 1 << 15,
         sh_none   = 1 << 14,
         sh_read   = 1 << 13,
         sh_write  = 1 << 12
         };

      nfstream(): fstream() {};
      nfstream(const signed char *name, int mode, int prot =
filebuf::openprot);
      nfstream(const unsigned char *name, int mode, int prot =
filebuf::openprot);
      nfstream(int fd): fstream(fd) {};
      nfstream(int fd, char *buffer, int mode)
         : fstream(fd, buffer, mode) {};
      ~nfstream() { close(); }

      void open(const signed char *name, int mode, int prot =
filebuf::openprot);
      void open(const unsigned char *name, int mode, int prot =
filebuf::openprot)
         {
         open((const signed char *) name, mode, prot);
         };
      void close();

   private:
      int fd;
   };

class nifstream: public ifstream
   {
   public:
      enum
         {
         sh_compat = 1 << 15,
         sh_none   = 1 << 14,
         sh_read   = 1 << 13,
         sh_write  = 1 << 12
         };

      nifstream(): ifstream() {};
      nifstream(const signed char *name, int mode, int prot =
filebuf::openprot);
      nifstream(const unsigned char *name, int mode, int prot =
filebuf::openprot);
      nifstream(int fd): ifstream(fd) {};
      nifstream(int fd, char *buffer, int mode)
         : ifstream(fd, buffer, mode) {};
      ~nifstream() { close(); }

      void open(const signed char *name, int mode, int prot =
filebuf::openprot);
      void open(const unsigned char *name, int mode, int prot =
filebuf::openprot)
         {
         open((const signed char *) name, mode, prot);
         };
      void close();

   private:
      int fd;
   };

class nofstream: public ofstream
   {
   public:
      enum
         {
         sh_compat = 1 << 15,
```

```
101 |          sh_none   = 1 << 14,
102 |          sh_read   = 1 << 13,
103 |          sh_write  = 1 << 12
104 |          };
105 |
106 |       nofstream(): ofstream() {};
107 |       nofstream(const signed char *name, int mode, int prot =
108 | filebuf::openprot);
109 |       nofstream(const unsigned char *name, int mode, int prot =
110 | filebuf::openprot);
111 |       nofstream(int fd): ofstream(fd) {};
112 |       nofstream(int fd, char *buffer, int mode)
113 |          : ofstream(fd, buffer, mode) {};
114 |       ~nofstream() { close(); }
115 |
116 |       void open(const signed char *name, int mode, int prot =
117 | filebuf::openprot);
118 |       void open(const unsigned char *name, int mode, int prot =
119 | filebuf::openprot)
120 |          {
121 |          open((const signed char *) name, mode, prot);
122 |          };
123 |       void close();
124 |
125 |    private:
126 |       int fd;
127 |    };
128 |
129 | /* --- Prototypes ---------------------------------------------------- */
130 | /* --- External Variables -------------------------------------------- */
131 |
132 | #endif
133 |
134 | /* --- End of File --------------------------------------------------- */
```

TEXT STATISTICS

```
  3978 characters
   134 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
    24 constants [numeric]
```

SYMBOL TABLE

```
    __NETSTREAM_H           16      17
    buffer          44      45      78      79     112     113
    close           46      55      80      89     114     123
    defined         16
    fd              43+     44      45      58      77+     78      79      92     111+    112     113
           126
    filebuf         40      42      49      51      74      76      83      85     108     110     117
           119
    fstream         20      27      38      43      45
    h       20      21
    ifstream        61      72      77      79
    mode            39      41      44      45      48      50      53      73      75      78      79
            82      84      87     107     109     112     113     116     118     121
    name            39      41      48      50      53      73      75      82      84      87     107
           109     116     118     121
    nfstream        27      38      39      41      43      44      46
    nifstream       61      72      73      75      77      78      80
    nofstream       95     106     107     109     111     112     114
    ofstream        95     106     111     113
    open            48      50      53      82      84      87     116     118     121
    openprot        40      42      49      51      74      76      83      85     108     110     117
           119
    prot            39      41      48      50      53      73      75      82      84      87     107
           109     116     118     121
    sh_compat       32      66     100
    sh_none         33      67     101
    sh_read         34      68     102
    sh_write        35      69     103
    share           21
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  NUMCNVRT.H - Header file for SNIPPETS numerical <=> string conversions
*/

#ifndef NUMCNVRT__H
#define NUMCNVRT__H

#include <stddef.h>     /* For size_t     */
#include "sniptype.h"
#include "round.h"
#include "pi.h"

#define R_ERROR -2                              /* EVAL.C Range error   */

/*
**  Callable library functions begin here
*/

char  * base_convert(const char *in, char *out,
             size_t len, int rin, int rout);           /* Bascnvrt.C     */
char  * comma_float(double num, char *buf, int dec);  /* Commaflt.C     */
size_t  commafmt(char *buf, int bufsize, long N);     /* Commafmt.C     */
char  * eng(double value, int places);                /* Eng.C          */
int     evaluate(char *line, double *val);            /* Eval.C         */
char  * fmt_money(double amt);                        /* Fmtmoney.C     */
long    hexorint(const char *string);                 /* Hexorint.C     */

char  * ltostr(long num, char *string,
            size_t max_chars, unsigned base);      /* Ltostr.C       */

char  * ordinal_text(int number);                     /* Ord_Text.C     */
int     scanfrac (const char buf[], double *f);       /* Scanfrac.C     */

unsigned int hstr_i(char *cptr);                      /* Hstr_I.C       */

char *long2roman(long val, char *buf, size_t buflen); /* L2Roman.C      */
long  roman2long(const char *str);                    /* Roman2L.C      */


#if defined(__ZTC__) && !defined(__SC__)
 char * ltoa(long val, char *buf, int base);          /* Ltoa.C         */
#endif


/*
**  File: STR27SEG.C
*/

struct Seg7disp {
      unsigned seg_a : 1;
      unsigned seg_b : 1;
      unsigned seg_c : 1;
      unsigned seg_d : 1;
      unsigned seg_e : 1;
      unsigned seg_f : 1;
      unsigned seg_g : 1;
};

char *str27seg(char *string);


#endif /* NUMCNVRT__H */
```

TEXT STATISTICS

```
  1929 characters
    64 lines
```

LEXICAL STATISTICS

```
    21 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     3 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    N       24
    NUMCNVRT__H              7       8
    R_ERROR         15
    Seg7disp        51
    __SC__          42
    __ZTC__         42
    amt             27
    base            31      43
    base_convert            21
    buf             23      24      34      38      43
    buflen          38
    bufsize         24
    comma_float             23
    commafmt        24
    cptr            36
    dec             23
    defined         42+
    eng             25
    evaluate        26
    f       34
    fmt_money       27
    h       10
    hexorint        28
    hstr_i          36
    in              21
    len             22
    line            26
    long2roman              38
    ltoa            43
    ltostr          30
    max_chars       31
    num             23      30
    number          33
    ordinal_text            33
    out             21
    places          25
    rin             22
    roman2long              39
    rout            22
    scanfrac        34
    seg_a           52
    seg_b           53
    seg_c           54
    seg_d           55
    seg_e           56
    seg_f           57
    seg_g           58
    size_t          22      24      31      38
    stddef          10
    str             39
    str27seg        61
    string          28      30      61
    val             26      38      43
    value           25
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  NWLINNAME.C - Novell Netware: getNwLoginName
**  This should work for Advanced NetWare (2.x) -> NetWare386 3.x & 4.x
**
**  Written by david nugent, public domain
*/

/* #define TEST */            /* Uncomment to compile test main()       */

#include <string.h>
#include <process.h>
#include <dos.h>
#include <errno.h>
#include "snpdosys.h"

#define addrOffset(val)     (unsigned short)(((long)(val)) & 0xfff)
    /* Invoke Novell Netware API.  */
#define callNetware(func, reqbuf, repbuf, inregs, segregs) \
    { \
        (inregs)->h.ah = (func); \
        (inregs)->x.si = addrOffset(reqbuf); \
        (inregs)->x.di = addrOffset(repbuf); \
        intdosx(inregs, inregs, segregs); \
    }

#if defined(_MSC_VER) || defined(_QC) || defined(__WATCOM__)
    #pragma pack(1)
#elif defined(__ZTC__)
    #pragma ZTC align 1
#elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
    #pragma option -a-
#endif

/*
**  Get Connection Information E3(16)
*/

struct _conninfo
{
    unsigned short len;
    unsigned char func;
    unsigned char cno;
};

struct _connrep
{
    unsigned short len;
    unsigned long objectID;
    unsigned short objecttype;
    char objectname[48];
    unsigned char logintime[7];
    unsigned char reserved[39];
};

#if defined(_MSC_VER) || defined(_QC) || defined(__WATCOM__)
    #pragma pack()
#elif defined(__ZTC__)
    #pragma ZTC align 2
#elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
    #pragma option -a
#endif

int getNwLoginName (char * namebuf)
{
      union REGS r;

      r.x.ax = 0xDC00;    /* Get Connection Number */
      intdos(&r, &r);

      /*
      ** If the connection number is in range 1-100 (ie. valid),
      ** get Connection Information and retrieve the user name.
      */

      if (r.h.al <= 0 ||  r.h.al > 100)
            return errno = EINVAL;

      else
      {
            struct SREGS s;
            static struct _connrep connrep;
            static struct _conninfo conninfo;

            conninfo.len = sizeof(conninfo) - sizeof(conninfo.len);
            conninfo.func = 0x16;   /* Get connection information */
            conninfo.cno = r.h.al;
            connrep.len = sizeof(connrep) - sizeof(connrep.len);
            segread(&s);
            s.es = s.ds;    /* ds:si=request buffer, es:di=reply buffer */

            /* login info */

            callNetware(0xE3, &conninfo, &connrep, &r, &s);
            if (r.h.al == 0)
            {
                  strcpy (namebuf, connrep.objectname);
                  return 0;
            }
```

```
 101 |             return r.h.al;
 102 |       }
 103 | }
 104 |
 105 | #if defined(TEST)
 106 |
 107 | #include <stdio.h>
 108 |
 109 | int
 110 | main()
 111 | {
 112 |       char loginname[48];
 113 |       int rc = getNwLoginName (loginname);
 114 |
 115 |       if (rc == 0)
 116 |             printf ("Your login name is '%s'\n", loginname);
 117 |       else  printf ("Error %d calling Netware, %s\n", rc, strerror(errno));
 118 |       return 0;
 119 | }
 120 |
 121 | #endif /* TEST */
```

TEXT STATISTICS

```
  3037 characters
   121 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    24 preprocessor instructions
     0 constants [character]
     3 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    EINVAL          78
    REGS            67
    SREGS           82
    TEST           105
    ZTC             31      60
    _MSC_VER        28      57
    _QC             28      57
    __TURBOC__              32+     61+
    __WATCOM__              28      57
    __ZTC__         30      59
    _conninfo       40      84
    _connrep        47      83
    a       33      62
    addrOffset              18      23      24
    ah              22
    al              77+     88      96     101
    align           31      60
    ax              69
    callNetware             20      95
    cno             44      88
    conninfo        84      86+     87      88      95
    connrep         83      89+     95      98
    defined         28+     30      32      57+     59      61     105
    di              24
    dos             14
    ds              91
    errno           15      78     117
    es              91
    func            20      22      43      87
    getNwLoginName          65     113
    h       12      13      14      15      22      77+     88      96     101     107
    inregs          20      22      23      24      25+
    intdos          70
    intdosx         25
    len             42      49      86+     89+
    loginname      112     113     116
    logintime       53
    main           110
    namebuf         65      98
    objectID        50
    objectname              52      98
    objecttype              51
    option          33      62
    pack            29      58
    printf         116     117
    process         13
    r       67      69      70+     77+     88      95      96     101
    rc             113     115     117
    repbuf          20      24
    reqbuf          20      23
    reserved        54
    s       82      90      91+     95
    segread         90
    segregs         20      25
    si              23
    stdio          107
    strcpy          98
    strerror       117
    string          12
    val             18+
    x       23      24      69
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  OPSYS.C  Operating environment detection and time-slicing support for
 *           MS-DOS applications.
 *
 *  Written in June 1996 by Andrew Clarke and released to the public domain.
 */

#include <dos.h>
#include "opsys.h"

int opsys_id = OS_MSDOS;

int opsysDetect(void)
{
#if defined(__TURBOC__)
    /* detect DESQview */
    _AX = 0x2B01;
    _CX = 0x4445;
    _DX = 0x5351;
    geninterrupt(0x21);
    if (_AL != 0xFF)
    {
        opsys_id = OS_DESQVIEW;
        return opsys_id;
    }

    /* detect OS/2 */
    _AH = 0x30;
    geninterrupt(0x21);
    if (_AL >= 10)
    {
        opsys_id = OS_OS2;
        return opsys_id;
    }

    /* detect Windows */
    _AX = 0x160A;
    geninterrupt(0x2F);
    if (_AX == 0)
    {
        opsys_id = OS_WINDOWS;
        return opsys_id;
    }
#else
    union REGS regs;

    /* detect DESQview */
#if defined(__WATCOMC__) && defined(__386__)
    regs.w.ax = 0x2B01;
    regs.w.cx = 0x4445;
    regs.w.dx = 0x5351;
    int386(0x21, &regs, &regs);
#else
    regs.x.ax = 0x2B01;
    regs.x.cx = 0x4445;
    regs.x.dx = 0x5351;
#ifdef __EMX__
    _int86(0x21, &regs, &regs);
#else
    int86(0x21, &regs, &regs);
#endif
#endif
    if (regs.h.al != 0xFF)
    {
        opsys_id = OS_DESQVIEW;
        return opsys_id;
    }

    /* detect OS/2 */
    regs.h.ah = 0x30;
#if defined(__WATCOMC__) && defined(__386__)
    int386(0x21, &regs, &regs);
#else
#ifdef __EMX__
    _int86(0x21, &regs, &regs);
#else
    int86(0x21, &regs, &regs);
#endif
#endif
    if (regs.h.al >= 10)
    {
        opsys_id = OS_OS2;
        return opsys_id;
    }

    /* detect Windows */
#if defined(__WATCOMC__) && defined(__386__)
    regs.w.ax = 0x160A;
    int386(0x2F, &regs, &regs);
    if (regs.w.ax == 0)
    {
        opsys_id = OS_WINDOWS;
        return opsys_id;
    }
#else
    regs.x.ax = 0x160A;
#ifdef __EMX__
    _int86(0x2F, &regs, &regs);
```

```
#else
    int86(0x2F, &regs, &regs);
#endif
    if (regs.x.ax == 0)
    {
        opsys_id = OS_WINDOWS;
        return opsys_id;
    }
#endif
#endif

    /* must be MS-DOS */
    opsys_id = OS_MSDOS;
    return opsys_id;
}

void opsysTimeSlice(void)
{
#if defined(__TURBOC__)
    switch (opsys_id)
    {
    case OS_MSDOS:
        geninterrupt(0x28);
        break;
    case OS_OS2:
    case OS_WINDOWS:
        _AX = 0x1680;
        geninterrupt(0x2F);
        break;
    case OS_DESQVIEW:
        _AX = 0x1000;
        geninterrupt(0x15);
        break;
    case OS_DBLDOS:
        _AX = 0xEE01;
        geninterrupt(0x21);
        break;
    case OS_NETWARE:
        _BX = 0x000A;
        geninterrupt(0x7A);
        break;
    default:
        break;
    }
#else
    union REGS regs;
    switch (opsys_id)
    {
    case OS_MSDOS:
#if defined(__WATCOMC__) && defined(__386__)
        int386(0x28, &regs, &regs);
#else
#ifdef __EMX__
        _int86(0x28, &regs, &regs);
#else
        int86(0x28, &regs, &regs);
#endif
#endif
        break;

    case OS_OS2:
    case OS_WINDOWS:
#if defined(__WATCOMC__) && defined(__386__)
        regs.w.ax = 0x1680;
        int386(0x2F, &regs, &regs);
#else
        regs.x.ax = 0x1680;
#ifdef __EMX__
        _int86(0x2F, &regs, &regs);
#else
        int86(0x2F, &regs, &regs);
#endif
#endif
        break;

    case OS_DESQVIEW:
#if defined(__WATCOMC__) && defined(__386__)
        regs.w.ax = 0x1000;
        int386(0x15, &regs, &regs);
#else
        regs.x.ax = 0x1000;
#ifdef __EMX__
        _int86(0x15, &regs, &regs);
#else
        int86(0x15, &regs, &regs);
#endif
#endif
        break;

    case OS_DBLDOS:
#if defined(__WATCOMC__) && defined(__386__)
        regs.w.ax = 0xEE01;
        int386(0x21, &regs, &regs);
#else
        regs.x.ax = 0xEE01;
#ifdef __EMX__
        _int86(0x21, &regs, &regs);
#else
        int86(0x21, &regs, &regs);
#endif
```

```
#endif
        break;

    case OS_NETWARE:
#if defined(__WATCOMC__) && defined(__386__)
        regs.w.bx = 0x000A;
        int386(0x7A, &regs, &regs);
#else
        regs.x.bx = 0x000A;
#ifdef __EMX__
        _int86(0x7A, &regs, &regs);
#else
        int86(0x7A, &regs, &regs);
#endif
#endif
        break;

    default:
        break;
    }
#endif
}

#ifndef __EMX__

unsigned short opsysGetVideoSeg(unsigned short assumed_base)
{
    if (opsys_id == OS_DESQVIEW)
    {
#if defined(__TURBOC__)
        _ES = assumed_base;
        _DI = 0;
        _AX = 0xFE00;
        geninterrupt(0x10);
        return _ES;
#else
        union REGS regs;
        struct SREGS sregs;
        sregs.es = assumed_base;
#if defined(__WATCOMC__) && defined(__386__)
        regs.w.di = 0;
        regs.w.ax = 0xFE00;
        int386x(0x10, &regs, &regs, &sregs);
#else
        regs.x.di = 0;
        regs.x.ax = 0xFE00;
        int86x(0x10, &regs, &regs, &sregs);
#endif
        return sregs.es;
#endif
    }
    return assumed_base;
}

#endif
```

```
TEXT STATISTICS

  5106 characters
   255 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
    64 preprocessor instructions
     0 constants [character]
     1 constants [string]
    74 constants [numeric]

SYMBOL TABLE

    OS_DBLDOS      134     190
    OS_DESQVIEW             25      67     130     176     228
    OS_MSDOS        13     113     122     149
    OS_NETWARE              138     204
    OS_OS2          34      84     125     161
    OS_WINDOWS              43      94     106     126     162
    REGS            47     146     237
    SREGS          238
    _AH             30
    _AL             23      32
    _AX             19      39      41     127     131     135     233
    _BX            139
    _CX             20
    _DI            232
    _DX             21
    _ES            231     235
    __386__         50      73      89     150     163     177     191     205     240
    __EMX__         59      76      99     153     168     182     196     210     224
    __TURBOC__              17     119     230
    __WATCOMC__             50      73      89     150     163     177     191     205     240
    _int86          60      77     100     154     169     183     197     211
    ah              72
    al              65      82
    assumed_base            226     231     239     252
    ax              51      56      90      92      98     104     164     167     178     181     192
           195     242     246
    bx             206     209
    cx              52      57
    defined         17      50+     73+     89+    119     150+    163+    177+    191+    205+    230
           240+
    di             241     245
    dos             10
    dx              53      58
    es             239     249
    geninterrupt            22      31      40     123     128     132     136     140     234
    h       10      65      72      82
    int386          54      74      91     151     165     179     193     207
    int386x        243
    int86           62      79     102     156     171     185     199     213
    int86x         247
    opsysDetect             15
    opsysGetVideoSeg        226
    opsysTimeSlice          117
    opsys_id        13      25      26      34      35      43      44      67      68      84      85
           94      95     106     107     113     114     120     147     228
    regs            47      51      52      53      54+     56      57      58      60+     62+     65
           72      74+     77+     79+     82      90      91+     92      98     100+    102+    104
          146     151+    154+    156+    164     165+    167     169+    171+    178     179+    181
          183+    185+    192     193+    195     197+    199+    206     207+    209     211+    213+
          237     241     242     243+    245     246     247+
    sregs          238     239     243     247     249
    w       51      52      53      90      92     164     178     192     206     241     242
    x       56      57      58      98     104     167     181     195     209     245     246
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 |  *  OPSYS.H  Operating environment detection and time-slicing support for
  5 |  *           MS-DOS applications.
  6 |  *
  7 |  *  Written in June 1996 by Andrew Clarke and released to the public domain.
  8 |  */
  9 |
 10 | #ifndef __OPSYS_H__
 11 | #define __OPSYS_H__
 12 |
 13 | #define OS_MSDOS     0  /* MS-DOS */
 14 | #define OS_OS2       1  /* IBM OS/2 */
 15 | #define OS_WINDOWS   2  /* Microsoft Windows */
 16 | #define OS_DESQVIEW  3  /* Quarterdeck DESQview */
 17 | #define OS_DBLDOS    4  /* DoubleDOS */
 18 | #define OS_NETWARE   5  /* Novell Netware */
 19 |
 20 | extern int opsys_id;
 21 |
 22 | int opsysDetect(void);
 23 | void opsysTimeSlice(void);
 24 | #ifndef __EMX__
 25 | unsigned short opsysGetVideoSeg(unsigned short assumed_base);
 26 | #endif
 27 |
 28 | #endif
```

TEXT STATISTICS

```
   733 characters
    28 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     0 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    OS_DBLDOS       17
    OS_DESQVIEW             16
    OS_MSDOS        13
    OS_NETWARE              18
    OS_OS2          14
    OS_WINDOWS              15
    __EMX__         24
    __OPSYS_H__             10      11
    assumed_base            25
    opsysDetect             22
    opsysGetVideoSeg        25
    opsysTimeSlice          23
    opsys_id        20
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Originally published as part of the MicroFirm Function Library
  5 | **
  6 | **  Copyright 1991, Robert B.Stout
  7 | **
  8 | **  With modifications suggested by Maynard Hogg
  9 | **
 10 | **  The user is granted a free limited license to use this source file
 11 | **  to create royalty-free programs, subject to the terms of the
 12 | **  license restrictions specified in the LICENSE.MFL file.
 13 | **
 14 | **  Function to return ordinal text.
 15 | */
 16 |
 17 | #include "numcnvrt.h"
 18 |
 19 | static char *text[] = {"th", "st", "nd", "rd"};
 20 |
 21 | char *ordinal_text(int number)
 22 | {
 23 |       if (((number %= 100) > 9 && number < 20) || (number %= 10) > 3)
 24 |             number = 0;
 25 |       return text[number];
 26 | }
 27 |
 28 | #ifdef TEST
 29 |
 30 | #include <stdio.h>
 31 |
 32 | main()
 33 | {
 34 |       int i;
 35 |
 36 |       for (i = 0; i < 26; ++i)
 37 |             printf("%d%s\n", i, ordinal_text(i));
 38 |       return 0;
 39 | }
 40 |
 41 | #endif /* TEST */
```

TEXT STATISTICS

```
   889 characters
    41 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     6 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    TEST            28
    h       30
    i       34      36+     37+
    main            32
    number          21      23+     24      25
    ordinal_text            21      37
    printf          37
    stdio           30
    text            19      25
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  OS/2 Reboot function
**
**  public domain by Mark Kimes
*/

#if defined(__MSDOS__) || defined(MSDOS)
 #error OS2_BOOT.C is NOT intended for DOS programs ...
#endif

#define INCL_DOS

#include <os2.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
      HFILE  hf;
#ifdef __32BIT__
      ULONG  dummy;
      ULONG  rc;
#else
      USHORT dummy;
      USHORT rc;
#endif

      rc = DosOpen("DOS$",&hf,&dummy,0L,FILE_NORMAL,FILE_OPEN,
            OPEN_ACCESS_WRITEONLY | OPEN_SHARE_DENYNONE |
            OPEN_FLAGS_FAIL_ON_ERROR,NULL);
      if(!rc)
      {
            /* Flush the caches           */

            DosShutdown(1);

            /* Shut down the file system  */

            DosShutdown(1);
            rc = DosShutdown(0);
            if(!rc)
            {
                  /* reboot machine */

#ifdef __32BIT__
                  rc = DosDevIOCtl(hf,0xd5,0xab,NULL,0,NULL,NULL,0,NULL);
#else
                  rc = DosDevIOCtl(NULL,NULL,0xab,0xd5,hf);
#endif
                  DosClose(hf);
            }
      }
      return rc;
}
```

TEXT STATISTICS

```
  1182 characters
    56 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     1 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    C       10
    DOS             10
    DosClose        52
    DosDevIOCtl             48      50
    DosOpen         30
    DosShutdown             37      41      42
    FILE_NORMAL             30
    FILE_OPEN       30
    HFILE           21
    INCL_DOS        13
    MSDOS            9
    NOT             10
    NULL            32      48+     50+
    OPEN_ACCESS_WRITEONLY           31
    OPEN_FLAGS_FAIL_ON_ERROR        32
    OPEN_SHARE_DENYNONE             31
    OS2_BOOT        10
    ULONG           23      24
    USHORT          26      27
    __32BIT__       22      47
    __MSDOS__        9
    defined          9+
    dummy           23      26      30
    h       15      16      17
    hf              21      30      48      50      52
    intended        10
    is              10
    main            19
    os2             15
    programs        10
    rc              24      27      30      33      42      43      48      50      55
    stdio           17
    stdlib          16
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  OS_ID.C
**
**  Based upon public domain works by David Gibbs & Stephen Lindholm
*/

#define OS_ID_MAIN
#include "snpdosys.h"
#include <dos.h>

struct i_os_ver id_os_ver[TOT_OS];
int id_os_type;
int id_os;

const char *id_os_name[TOT_OS] = {
      "DOS",
      "OS/2 DOS",
      "DESQview",
      "Windows Std",
      "Windows 386"
      };

/*
**  get_os() - Determine OS in use
*/

int get_os (void)
{
      union REGS t_regs;
      int osmajor, osminor;
      unsigned status;

      id_os_type = 0;
      id_os = 0;

      /* test for DOS or OS/2 */

      t_regs.h.ah = 0x30;
      int86(0x21, &t_regs, &t_regs);
      osmajor = t_regs.h.al;
      osminor = t_regs.h.ah;

      if (osmajor < 10)
      {
            id_os_ver[DOS].maj = osmajor;
            id_os_ver[DOS].min = osminor;
            id_os_type = id_os_type | is_DOS;
      }
      else
      {
            /* OS/2 v1.x DOS Box returns 0x0A */

            id_os_type = id_os_type | is_OS2;

            /* OS/2 v2.x DOS Box returns 0x14 */

            id_os_ver[OS2].maj = osmajor/10;
            id_os_ver[OS2].min = osminor;
      }

      /* test for Windows */

      t_regs.x.ax = 0x1600;         /* check enhanced mode operation    */
      int86(0x2F, &t_regs, &t_regs);
      status = t_regs.h.al;

      if ((0x00 == status) || (0x80 == status))
      {
            /*
            ** Can't trust it...
            **  let's check if 3.1 is running in standard mode or what?
            */

            t_regs.x.ax = 0x160A;
            int86( 0x2F, &t_regs, &t_regs );
            if (0 == t_regs.x.ax)
            {
                  id_os_ver[WINS].maj = t_regs.h.bh;
                  id_os_ver[WINS].min = t_regs.h.bl;
                  id_os_type = id_os_type | is_WINS;
            }
      }
      else if ((0x01 == status) || (0xff == status))
      {
            id_os_ver[WINS].maj = 2;
            id_os_ver[WINS].min = 1;
            id_os_type = id_os_type | is_WINS;
      }
      else
      {
            id_os_ver[WINS].maj = t_regs.h.al;
            id_os_ver[WINS].min = t_regs.h.ah;
            id_os_type = id_os_type | is_WINS;
      }

      /* Test for DESQview */

      t_regs.x.cx = 0x4445;               /* load incorrect date */
```

```
101 |       t_regs.x.dx = 0x5351;
102 |       t_regs.x.ax = 0x2B01;               /*  DV set up call     */
103 |
104 |       intdos(&t_regs, &t_regs);
105 |       if (t_regs.h.al != 0xFF)
106 |       {
107 |             id_os_type = id_os_type | is_DV;
108 |             id_os_ver[DV].maj = t_regs.h.bh;
109 |             id_os_ver[DV].min = t_regs.h.bl;
110 |       }
111 |
112 |       if (id_os_type & is_DOS)
113 |             id_os = DOS;
114 |       if (id_os_type & is_WINS)
115 |             id_os = WINS;
116 |       if (id_os_type & is_WIN3)
117 |             id_os = WIN3;
118 |       if (id_os_type & is_DV)
119 |             id_os = DV;
120 |       if (id_os_type & is_OS2)
121 |             id_os = OS2;
122 |
123 |       return(id_os);
124 | }
125 |
126 | /*
127 | **  Give up a time slice to the OS
128 | */
129 |
130 | void t_slice(int t_os)
131 | {
132 |       union REGS t_regs;
133 |
134 |       switch (t_os)
135 |       {
136 |       case DOS  :
137 |             break;
138 |
139 |       case OS2  :
140 |       case WIN3 :
141 |       case WINS :
142 |             t_regs.x.ax = 0x1680;
143 |             int86(0x2f,&t_regs,&t_regs);
144 |             break;
145 |
146 |       case DV   :
147 |             t_regs.x.ax = 0x1000;
148 |             int86(0x15,&t_regs,&t_regs);
149 |             break;
150 |       } /* switch(t_os) */
151 | }
152 |
153 |
154 | #ifdef TEST
155 |
156 | #include <stdio.h>
157 |
158 | int main(void)
159 | {
160 |       int ostype = get_os();
161 |
162 |       printf("%s version %d.%d\n",
163 |             id_os_name[ostype],
164 |             id_os_ver[ostype].maj,
165 |             id_os_ver[ostype].min);
166 |
167 |       return(0);
168 | }
169 |
170 | #endif /* TEST */
```

TEXT STATISTICS

```
  3789 characters
   170 lines
```

LEXICAL STATISTICS

```
    15 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     7 constants [string]
    26 constants [numeric]
```

SYMBOL TABLE

```
    DOS             47      48     113     136
    DV             108     109     119     146
    OS2             59      60     121     139
    OS_ID_MAIN               9
    REGS            31     132
    TEST           154
    TOT_OS          13      17
    WIN3           117     140
    WINS            80      81      87      88      93      94     115     141
    ah              40      43      94
    al              42      67      93     105
    ax              65      76      78     102     142     147
    bh              80     108
    bl              81     109
    cx             100
    dos             11
    dx             101
    get_os          29     160
    h       11      40      42      43      67      80      81      93      94     105     108     109
           156
    i_os_ver        13
    id_os           15      36     113     115     117     119     121     123
    id_os_name              17     163
    id_os_type              14      35      49+     55+     82+     89+     95+    107+    112     114
           116     118     120
    id_os_ver       13      47      48      59      60      80      81      87      88      93      94
           108     109     164     165
    int86           41      66      77     143     148
    intdos         104
    is_DOS          49     112
    is_DV          107     118
    is_OS2          55     120
    is_WIN3        116
    is_WINS         82      89      95     114
    main           158
    maj             47      59      80      87      93     108     164
    min             48      60      81      88      94     109     165
    osmajor         32      42      45      47      59
    osminor         32      43      48      60
    ostype         160     163     164     165
    printf         162
    status          33      67      69+     85+
    stdio          156
    t_os           130     134
    t_regs          31      40      41+     42      43      65      66+     67      76      77+     78
           80      81      93      94     100     101     102     104+    105     108     109     132
          142     143+    147     148+
    t_slice        130
    x       65      76      78     100     101     102     142     147
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /**/char q='"',*a="*//**/char q='%c',*a=%c%s%c*/};)b(stup;]d[b=]d-472[b)--d(elihw;)q,a,q,q,2+a,b(ftnir
     |  ps;)b(stup{)(niam;731=d tni;]572[b,"
   4 | ,b[275];int d=137;main(){puts(b);sprintf(b,a+2,q,q,a,q);while(d--)b[274-d]=b[d];puts(b);}/*c%s%c%=a*,'
     |  c%'=q rahc/**//*"=a*,'"'=q rahc/**/
```

TEXT STATISTICS

```
   323 characters
     4 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     1 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

a	3	4+
b	4+	
d	4+	
main		4
puts		4+
q	3	4+
sprintf		4

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PALNFILT.C - A palindrome filter.
**
**  Reads lines of text and only passes lines which are palindromes, i.e.
**  those which read the same forwards or backwards. Includes options switches
**  to ignore case and/or punctuation, e.g.
**
**  Input               Output      w/ -C       w/ -P       w/ -C -P
**  ---------------     ----------  ----------  ----------  ---------------
**  toot                toot        toot        toot        toot
**  Toot                            Toot                    Toot
**  toot!                                       toot!       toot!
**  Madam, I'm Adam                                         Madam, I'm Adam
**  This is a test.
**  ------------------------------------------------------------------------
**
**  public domain demo by Bob Stout
**
**  Uses GETOPTS.C and CANT.C, also from SNIPPETS
*/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "getopts.h"
#include "snipfile.h"                   /* For cant()           */

#define LAST_CHAR(s) (((char *)s)[strlen(s) - 1])
#define NUL '\0'

void usage(int exitval);
char *rmpunc(char *str);

Boolean_T fold = False_, punc = False_, help = False_;

struct Option_Tag options[] = {
      {'c', False_, Boolean_Tag, &fold, NULL, NULL, NULL},
      {'p', False_, Boolean_Tag, &punc, NULL, NULL, NULL},
      {'h', False_, Boolean_Tag, &help, NULL, NULL, NULL},
      { 0 , False_, Error_Tag  , NULL,  NULL, NULL, NULL}
};

/*
**  Implemented as a true filer, defaulting to stdin and stdout.
*/

main(int argc, char *argv[])
{
      FILE *infile = stdin, *outfile = stdout;
      char line[2][256];                        /* Nice & roomy   */

      if (Error_ == getopts(argc, argv))
            usage(-1);
      if (help)
            usage(0);
      if (1 < xargc)
            infile = cant(xargv[1], "r");
      if (2 < xargc)
            outfile = cant(xargv[2], "w");
      while (NULL != fgets(line[0], 255, infile))
      {
            char *p1, *p2;
            int OK;

            strcpy(line[1], line[0]);
            if ('\n' == LAST_CHAR(line[1]))
                  LAST_CHAR(line[1]) = NUL;
            if (fold)
                  strupr(line[1]);
            if (punc)
                  rmpunc(line[1]);
            for (p1 = line[1], p2 = &LAST_CHAR(line[1]), OK = 1;
                  p2 > p1; ++p1, --p2)
            {
                  if (*p1 != *p2)
                  {
                        OK = 0;
                        break;
                  }
            }
            if (OK)
                  fputs(line[0], outfile);
      }
      return 0;
}

/*
**  Tell 'em how it works
*/

void usage(int exitval)
{
      puts("Usage: PALNFILT [-cph] [infile] [outfile]");
      puts("where: h = Provide help (this display)");
      puts("       c = Ignore case");
      puts("       p = Ignore punctuation");
      puts("       infile defaults to stdin");
      puts("       outfile defaults to stdout");
```

```
 101 |       puts("notes: switches may not be concatenated but may be any case");
 102 |       exit(exitval);
 103 | }
 104 |
 105 | /*
 106 | **  Remove all punctuation from a string
 107 | */
 108 |
 109 | char *rmpunc(char *str)
 110 | {
 111 |       char *obuf, *nbuf;
 112 |
 113 |       for (obuf = str, nbuf = str; *obuf && obuf; ++obuf)
 114 |       {
 115 |             if (isalpha(*obuf))
 116 |                   *nbuf++ = *obuf;
 117 |       }
 118 |       *nbuf = NUL;
 119 |       return str;
 120 | }
```

TEXT STATISTICS

```
  3492 characters
   120 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     5 constants [character]
    11 constants [string]
    24 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       36
    Boolean_Tag             39      40      41
    Error_          54
    Error_Tag       42
    FILE            51
    False_          36+     39      40      41      42
    LAST_CHAR       30      68      69      74
    NUL             31      69     118
    NULL            39+     40+     41+     42+     62
    OK              65      74      79      83
    Option_Tag              38
    argc            49      54
    argv            49      54
    cant            59      61
    ctype           26
    exit           102
    exitval         33      93     102
    fgets           62
    fold            36      39      70
    fputs           84
    getopts         54
    h       24      25      26
    help            36      41      56
    infile          51      59      62
    isalpha        115
    line            52      62      67+     68      69      71      73      74+     84
    main            49
    nbuf           111     113     116     118
    obuf           111     113+    115     116
    options         38
    outfile         51      61      84
    p1              64      74      75+     77
    p2              64      74      75+     77
    punc            36      40      72
    puts            95      96      97      98      99     100     101
    rmpunc          34      73     109
    s       30+
    stdin           51
    stdio           24
    stdout          51
    str             34     109     113+    119
    strcpy          67
    string          25
    strlen          30
    strupr          71
    usage           33      55      57      93
    xargc           58      60
    xargv           59      61
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PARITY.C - Computes even or odd parity for various integral types
**
**  public domain demo by Bob Stout
*/

#include "parity.h"

unsigned parity32(unsigned long x, Parity_T even)
{
      x = x ^ (x >> 16);
      x = x ^ (x >> 8);
      x = x ^ (x >> 4);
      x = x ^ (x >> 2);
      x = x ^ (x >> 1);

      return ((unsigned)(x & 1)) ^ even;
}

unsigned parity16(unsigned short x, Parity_T even)
{
      x = x ^ (x >> 8);
      x = x ^ (x >> 4);
      x = x ^ (x >> 2);
      x = x ^ (x >> 1);

      return ((unsigned)(x & 1)) ^ even;
}

unsigned parity8(unsigned char x, Parity_T even)
{
      x = x ^ (x >> 4);
      x = x ^ (x >> 2);
      x = x ^ (x >> 1);

      return ((unsigned)(x & 1)) ^ even;
}

unsigned parity64(void *x, Parity_T even)
{
      union longlong *val64 = (union longlong *)x;

      return (parity32(val64->lo, even) ^ parity32(val64->hi, even));
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      while (--argc)
      {
            unsigned long n = strtoul(*(++argv), NULL, 10);

            printf("Even parity of %ld = %d\n", n, parity32(n, Even_));
            printf("Odd parity of  %ld = %d\n\n", n, parity32(n, Odd_));
      }
}

#endif
```

TEXT STATISTICS

```
  1329 characters
    64 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     3 constants [string]
    16 constants [numeric]
```

SYMBOL TABLE

```
    Even_           59
    NULL            57
    Odd_            60
    Parity_T        11      22      32      41
    TEST            48
    argc            53      55
    argv            53      57
    even            11      19      22      29      32      38      41      45+
    h       50      51
    hi              45
    lo              45
    longlong        43+
    main            53
    n       57      59+     60+
    parity16        22
    parity32        11      45+     59      60
    parity64        41
    parity8         32
    printf          59      60
    stdio           50
    stdlib          51
    strtoul         57
    val64           43      45+
    x       11      13+     14+     15+     16+     17+     19      22      24+     25+     26+     27+
            29      32      34+     35+     36+     38      41      43
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  PARITY.H
  5 | */
  6 |
  7 | typedef enum {Even_, Odd_} Parity_T;
  8 |
  9 | unsigned parity32(unsigned long x, Parity_T even);
 10 | unsigned parity16(unsigned short x, Parity_T even);
 11 | unsigned parity8(unsigned char x, Parity_T even);
 12 |
 13 | union longlong {
 14 |       unsigned long lo;
 15 |       unsigned long hi;
 16 | };
 17 |
 18 | unsigned parity64(void *x, Parity_T even);
```

TEXT STATISTICS

```
   384 characters
    18 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Even_            7
    Odd_             7
    Parity_T         7       9      10      11      18
    even             9      10      11      18
    hi              15
    lo              14
    longlong        13
    parity16        10
    parity32         9
    parity64        18
    parity8         11
    x        9      10      11      18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PARSDATE.C - A simple parser to extract dates from strings.
**
**  Original Copyright 1995 by Robert B. Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "datetime.h"

static unsigned  year_norml(unsigned year);
static Boolean_T get_fields(char *str);
static Boolean_T validate(unsigned year, unsigned month, unsigned day);
static Boolean_T isleap (unsigned yr);
static unsigned  get_month(char *month);

static char *fields[3];
static char *month[2][12] = {
      {"January", "February", "March", "April", "May", "June",
       "July", "August", "September", "October", "November", "December"},
      {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
       "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"}
};

/*
**  parse_date() - Parse a date string into its components
**
**  Arguments: 1 - String to parse
**             2 - Address of year storage
**             3 - Address of month storage
**             4 - Address of day storage
**             5 - Syntax to use
**
**  Returns: 0 For success, non-zero for range errors
**
**  Notes: The following syntaxes are supported:
**
**    Date ordering: USA (month, day, year),
**                   ISO (year, month, day)
**                   EUROPE (day, month, year)
**    Delimiters:    Spaces, commas, dashes, periods, slashes. (" ,-./")
**    Years:         2-digit years assumed to be between 1970 and 2069
**                   4-digit years required otherwise
*/

Boolean_T parse_date(const char *str,
                     unsigned   *year,
                     unsigned   *month,
                     unsigned   *day,
                     Syntax_T   syntax)
{
      unsigned yy, mm, dd;               /* Local data                */
      char str_copy[512];                /* Nice and roomy            */

      strcpy(str_copy, str);
      if (Error_ == get_fields(str_copy))
            return Error_;

      switch (syntax)
      {
      case USA:
            yy = atoi(fields[2]);
            dd = atoi(fields[1]);
            if (isalpha(*fields[0]))
            {
                  mm = get_month(fields[0]);
                  if (0 == mm)
                  {
                        return Error_;
                  }
            }
            else  mm = atoi(fields[0]);
            break;

      case ISO:
            yy = atoi(fields[0]);
            dd = atoi(fields[2]);
            if (isalpha(*fields[1]))
            {
                  mm = get_month(fields[1]);
                  if (0 == mm)
                        return Error_;
            }
            else  mm = atoi(fields[1]);
            break;

      case EUROPE:
            yy = atoi(fields[2]);
            dd = atoi(fields[0]);
            if (isalpha(*fields[1]))
            {
                  mm = get_month(fields[1]);
```

```
101 |                   if (0 == mm)
102 |                         return Error_;
103 |             }
104 |             else  mm = atoi(fields[1]);
105 |             break;
106 |
107 |       default:
108 |             return Error_;
109 |       }
110 |       yy = year_norml(yy);
111 |       if (Error_ == validate(yy, mm, dd))
112 |             return Error_;
113 |       *year  = yy;
114 |       *day   = dd;
115 |       *month = mm;
116 |       return Success_;
117 | }
118 |
119 | /*
120 | **  Utility function to split string fields
121 | */
122 |
123 | static Boolean_T get_fields(char *str)
124 | {
125 |       if (NULL == (fields[0] = strtok(str, " ,-./")))
126 |             return Error_;
127 |       if (NULL == (fields[1] = strtok(NULL, " ,-./")))
128 |             return Error_;
129 |       if (NULL == (fields[2] = strtok(NULL, " ,-./")))
130 |             return Error_;
131 |       if (NULL != strtok(NULL, " ,-./"))
132 |             return Error_;
133 |       return Success_;
134 | }
135 |
136 | /*
137 | **  Utility function to validate dates
138 | */
139 |
140 | static Boolean_T validate(unsigned year, unsigned month, unsigned day)
141 | {
142 |       unsigned int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
143 |
144 |       if (1 > month || 12 < month)
145 |             return Error_;
146 |       if (1 > day || day > (days[month - 1] + (2 == month && isleap(year))))
147 |             return Error_;
148 |       else  return Success_;
149 | }
150 |
151 | /*
152 | **  Utility function to detect leap years
153 | */
154 |
155 | static Boolean_T isleap (unsigned yr)
156 | {
157 |    return yr % 400 == 0 || (yr % 4 == 0 && yr % 100 != 0);
158 | }
159 |
160 | /*
161 | **  Utility function to normalize years
162 | */
163 |
164 | static unsigned year_norml(unsigned year)
165 | {
166 |       if (99 < year)
167 |             return year;
168 |       if (69 < year)
169 |             return year + 1900;
170 |       else  return year + 2000;
171 | }
172 |
173 | /*
174 | **  Utility function to determine month from string
175 | **
176 | **  (Note: Uses non-standard stricmp(), common to all DOS compilers
177 | */
178 |
179 | static unsigned get_month(char *str)
180 | {
181 |       int ix, iy;                        /* Indexes                    */
182 |
183 |       for (ix = 0; ix < 2; ++ix)
184 |       {
185 |             for (iy = 0; iy < 12; ++iy)
186 |             {
187 |                   if (Success_ == stricmp(str, month[ix][iy]))
188 |                         return iy + 1;
189 |             }
190 |       }
191 |       return 0;
192 | }
193 |
194 | #ifdef TEST
195 |
196 | #include <stdio.h>
197 |
198 | main(int argc, char *argv[])
199 | {
200 |       if (2 > argc)
```

```
201 |       {
202 |             puts("Usage: PARSDATE syntax date_string");
203 |             puts("       syntax = 0, 1, or 2");
204 |             return EXIT_FAILURE;
205 |       }
206 |       while (--argc)
207 |       {
208 |             Syntax_T syntax;
209 |             char *str;
210 |             unsigned yy, mm, dd;
211 |             Boolean_T retval;
212 |
213 |             if (4 > argc)
214 |                   break;
215 |             syntax = atoi(*++argv);
216 |             str = *++argv;
217 |             printf("str = \"%s\"\n", str);
218 |             retval = parse_date(str, &yy, &mm, &dd, syntax);
219 |             printf("parse_date(%d, \"%s\") returned %d\n"
220 |                   "  date = %u-%u-%u\n\n",
221 |                   syntax, str, retval, mm, dd, yy);
222 |       }
223 |       return EXIT_SUCCESS;
224 | }
225 |
226 | #endif /* TEST */
```

TEXT STATISTICS

```
  6180 characters
   226 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
    34 constants [string]
    60 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       20      21      22      54     123     140     155     211
    EUROPE          95
    EXIT_FAILURE           204
    EXIT_SUCCESS           223
    Error_          64      65      77      90     102     108     111     112     126     128     130
          132     145     147
    ISO             83
    NULL           125     127+    129+    131+
    Success_       116     133     148     187
    Syntax_T        58     208
    TEST           194
    USA             69
    argc           198     200     206     213
    argv           198     215     216
    atoi            70      71      80      84      85      92      96      97     104     215
    ctype           16
    day             21      57     114     140     146+
    days           142     146
    dd              60      71      85      97     111     114     210     218     221
    fields          25      70      71      72      74      80      84      85      86      88      92
           96       97      98     100     104     125     127     129
    get_fields              20      64     123
    get_month       23      74      88     100     179
    h       14      15      16     196
    isalpha         72      86      98
    isleap          22     146     155
    ix             181     183+    187
    iy             181     185+    187     188
    main           198
    mm              60      74      75      80      88      89      92     100     101     104     111
          115     210     218     221
    month           21      23      26      56     115     140     144+    146+    187
    parse_date              54     218
    printf         217     219
    puts           202     203
    retval         211     218     221
    stdio          196
    stdlib          14
    str             20      54      63     123     125     179     187     209     216     217     218
          221
    str_copy        61      63      64
    strcpy          63
    stricmp        187
    string          15
    strtok         125     127     129     131
    syntax          58      67     208     215     218     221
    validate        21     111     140
    year            19      21      55     113     140     146     164     166     167     168     169
          170
    year_norml              19     110     164
    yr              22     155     157+
    yy              60      70      84      96     110+    111     113     210     218     221
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PARSTIME.C - A simple parser to extract times from strings.
**
**  Public domain by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include "datetime.h"


/*
**  parse_time() - Parse a time string into its components
**
**  Arguments: 1 - String to parse
**             2 - Address of hours storage
**             3 - Address of minutes storage
**             4 - Address of seconds storage
**
**  Returns: 0 For success, non-zero for range errors
*/

Boolean_T parse_time(const char *str,
                     unsigned   *hours,
                     unsigned   *mins,
                     unsigned   *secs)
{
      unsigned hh, mm, ss;               /* Local data                */

      if (3 != sscanf((char *)str, "%u:%u:%u", &hh, &mm, &ss))
            return Error_;
      if (hh > 23 || mm > 59 || ss > 59)
            return Error_;
      *hours = hh;
      *mins  = mm;
      *secs  = ss;
      return Success_;
}

#ifdef TEST

main(int argc, char *argv[])
{
      if (2 > argc)
      {
            puts("Usage: PARSTIME time_string [...time_string]");
            return EXIT_FAILURE;
      }
      while (--argc)
      {
            char *str;
            unsigned hh, mm, ss;
            Boolean_T retval;

            retval = parse_time(str = *++argv, &hh, &mm, &ss);
            printf("parse_time(\"%s\") returned %d\n", str, retval);
            if (Success_ == retval)
                  printf("  time = %02u:%02u:%02u\n\n", hh, mm, ss);
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1635 characters
    65 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     5 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    Boolean_T       25      55
    EXIT_FAILURE            49
    EXIT_SUCCESS            62
    Error_          33      35
    Success_        39      59
    TEST            42
    argc            44      46      51
    argv            44      57
    h        9      10
    hh              30      32      34      36      54      57      60
    hours           26      36
    main            44
    mins            27      37
    mm              30      32      34      37      54      57      60
    parse_time              25      57
    printf          58      60
    puts            48
    retval          55      57      58      59
    secs            28      38
    ss              30      32      34      38      54      57      60
    sscanf          32
    stdio            9
    stdlib          10
    str             25      32      53      57      58
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**        A Pratt-Boyer-Moore string search, written by Jerry Coffin
**  sometime or other in 1991.  Removed from original program, and
**  (incorrectly) rewritten for separate, generic use in early 1992.
**  Corrected with help from Thad Smith, late March and early
**  April 1992...hopefully it's correct this time. Revised by Bob Stout.
**
**  This is hereby placed in the Public Domain by its author.
**
**  10/21/93 rdg  Fixed bug found by Jeff Dunlop
*/

#include <stddef.h>
#include <string.h>
#include <limits.h>

static size_t table[UCHAR_MAX + 1];
static size_t len;
static char *findme;

/*
**  Call this with the string to locate to initialize the table
*/

void init_search(const char *string)
{
      size_t i;

      len = strlen(string);
      for (i = 0; i <= UCHAR_MAX; i++)                    /* rdg 10/93 */
            table[i] = len;
      for (i = 0; i < len; i++)
            table[(unsigned char)string[i]] = len - i - 1;
      findme = (char *)string;
}

/*
**  Call this with a buffer to search
*/

char *strsearch(const char *string)
{
      register size_t shift;
      register size_t pos = len - 1;
      char *here;
      size_t limit=strlen(string);

      while (pos < limit)
      {
            while( pos < limit &&
                  (shift = table[(unsigned char)string[pos]]) > 0)
            {
                  pos += shift;
            }
            if (0 == shift)
            {
                  if (0 == strncmp(findme,
                        here = (char *)&string[pos-len+1], len))
                  {
                        return(here);
                  }
                  else  pos++;
            }
      }
      return NULL;
}

#ifdef TEST

#include <stdio.h>

main()
{
      char *here;
      char *find_strings[] = {"abb", "you", "not", "it", "dad", "yoo", "hoo",
                              "oo", "oh", "xx", "xx", "x", "x", NULL};
      char *search_strings[] = {"cabbie", "your", "It isn't here",
                                "But it is here", "hodad", "yoohoo", "yoohoo",
                                "yoohoo", "yoohoo", "yoohoo", "xx", "x", "."};
      int i;

      for (i = 0; find_strings[i]; i++)
      {
            init_search(find_strings[i]);
            here = strsearch(search_strings[i]);
            printf("\"%s\" is%s in \"%s\"", find_strings[i],
                  here ? "" : " not", search_strings[i]);
            if (here)
                  printf(" [\"%s\"]", here);
            putchar('\n');
      }

      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2673 characters
    98 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     1 constants [character]
    30 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

NULL		67	78							
TEST		70								
UCHAR_MAX		19	32							
find_strings			77	84	86	88				
findme		21	36	59						
h	15	16	17	72						
here		47	60	62	76	87	89	90	91	
i	29	32+	33	34+	35+	82	84+	86	87	88	89
init_search			27	86						
len		20	31	33	34	35	46	60+		
limit		48	50	52						
limits		17								
main		74								
pos		46	50	52	53	55	60	64		
printf		88	91							
putchar		92								
search_strings			79	87	89					
shift		45	53	55	57					
size_t		19	20	29	45	46	48			
stddef		15								
stdio		72								
string		16	27	31	35	36	43	48	53	60
strlen		31	48							
strncmp		59								
strsearch		43	87							
table		19	33	35	53					

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PCHWIO.C - SNIPPETS portable hardware I/O access under DOS
**
**  public domain by Bob Stout
*/

#include "pchwio.h"
#include "mk_fp.h"

#if defined(__ZTC__) && !defined(__SC__)

void FAR * getvect(unsigned intnum)
{
      unsigned seg, off;

      int_getvector(intnum, &off, &seg);
      return MK_FP(seg, off);
}

void setvect(unsigned intnum, void (INTERRUPT FAR *handler)())
{
      unsigned seg = FP_SEG(handler), off = FP_OFF(handler);

      int_setvector(intnum, off, seg);
}

#endif /* ZTC getvect(), setvect() */



#if defined(_MSC_VER) || defined(__WATCOMC__) || \
      defined(__ZTC__) || defined(__SC__)

#if !defined(MK_FP)
 #define MK_FP(seg,off) ((void far *)(((long)(seg) << 16)|(unsigned)(off)))
#endif

unsigned char Peekb(unsigned seg, unsigned ofs)
{
        unsigned char FAR *ptr;

        ptr = MK_FP(seg, ofs);
        return *ptr;
}

unsigned short Peekw(unsigned seg, unsigned ofs)
{
        unsigned FAR *ptr;

        ptr = MK_FP(seg, ofs);
        return *ptr;
}

void Pokeb(unsigned seg, unsigned ofs, unsigned char ch)
{
        unsigned char FAR *ptr;

        ptr = MK_FP(seg, ofs);
        *ptr = ch;
}

void Pokew(unsigned seg, unsigned ofs, unsigned short num)
{
        unsigned FAR *ptr;

        ptr = MK_FP(seg, ofs);
        *ptr = num;
}

#endif /* MSC/ZTC/WC Peek(), poke() */
```

TEXT STATISTICS

```
  1445 characters
    72 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     2 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    FAR             14      22      42      50      58      66
    FP_OFF          24
    FP_SEG          24
    INTERRUPT       22
    MK_FP           19      36      37      44      52      60      68
    Peekb           40
    Peekw           48
    Pokeb           56
    Pokew           64
    _MSC_VER        33
    __SC__          12      34
    __WATCOMC__             33
    __ZTC__         12      34
    ch              56      61
    defined         12+     33+     34+     36
    far             37
    getvect         14
    handler         22      24+
    int_getvector           18
    int_setvector           26
    intnum          14      18      22      26
    num             64      69
    off             16      18      19      24      26      37+
    ofs             40      44      48      52      56      60      64      68
    ptr             42      44      45      50      52      53      58      60      61      66      68
            69
    seg             16      18      19      24      26      37+     40      44      48      52      56
            60      64      68
    setvect         22
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PCHWIO.H - SNIPPETS header file for portable hardware I/O access under DOS
**
**  public domain by Bob Stout
*/

#ifndef PCHWIO__H
#define PCHWIO__H

#include <dos.h>
#include "extkword.h"


#if defined(__TURBOC__) || defined(__POWERC)
 #ifndef inp
  #define inp           inportb
 #endif
 #ifndef outp
  #define outp          outportb
 #endif
 #ifndef inpw
  #define inpw          inport
 #endif
 #ifndef outpw
  #define outpw         outport
 #endif
#elif defined(__ZTC__)
 #include <int.h>
 #define enable         int_on
 #define disable        int_off
 #if !defined(__SC__)
  void FAR * getvect(unsigned intnum);
  void setvect(unsigned intnum, void (INTERRUPT FAR *handler)());
 #else
  #define getvect       _dos_getvect
  #define setvect       _dos_setvect
 #endif
#else /* assume MSC/QC/WC */
 #include <conio.h>
 #if defined(__WATCOMC__)
  #include <i86.h>
 #endif
 #define enable         _enable
 #define disable        _disable
 #define getvect        _dos_getvect
 #define setvect        _dos_setvect
#endif


#if defined(_MSC_VER) || defined(__WATCOMC__) || \
      defined(__ZTC__) || defined(__SC__)

unsigned char  Peekb(unsigned seg, unsigned ofs);           /* PCHWIO.C */
unsigned short Peekw(unsigned seg, unsigned ofs);           /* PCHWIO.C */
void Pokeb(unsigned seg, unsigned ofs, unsigned char ch);   /* PCHWIO.C */
void Pokew(unsigned seg, unsigned ofs, unsigned short num); /* PCHWIO.C */

#elif defined(__TURBOC__)
 #define Peekw peek
 #define Pokew poke
 #define Peekb peekb
 #define Pokeb pokeb
#endif /* peek(), poke() */

#endif /* PCHWIO__H */
```

TEXT STATISTICS

1694 characters
67 lines

LEXICAL STATISTICS

9 comments [std-C]
0 comments [C++]
44 preprocessor instructions
0 constants [character]
1 constants [string]
0 constants [numeric]

SYMBOL TABLE

FAR	34	35					
INTERRUPT	35						
PCHWIO__H	9	10					
Peekb	55	63					
Peekw	56	61					
Pokeb	57	64					
Pokew	58	62					
_MSC_VER	52						
__POWERC	16						
__SC__	33	53					
__TURBOC__		16	60				
__WATCOMC__		42	52				
__ZTC__	29	53					
_disable	46						
_dos_getvect		37	47				
_dos_setvect		38	48				
_enable	45						
ch	57						
conio	41						
defined	16+	29	33	42	52+	53+	60
disable	32	46					
dos	12						
enable	31	45					
getvect	34	37	47				
h 12	30	41	43				
handler	35						
i86	43						
inp	17	18					
inport	24						
inportb	18						
inpw	23	24					
int_off	32						
int_on	31						
intnum	34	35					
num	58						
ofs	55	56	57	58			
outp	20	21					
outport	27						
outportb	21						
outpw	26	27					
peek	61						
peekb	63						
poke	62						
pokeb	64						
seg	55	56	57	58			
setvect	35	38	48				

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  demo code for converting Pascal strings to/from C strings
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #include <string.h>
 10 | #include "sniptype.h"
 11 |
 12 | #define P2Cconvert(s) do \
 13 |             {BYTE n = *(s); memmove((s), &(s)[1], n); s[n] = '\0';} while(0)
 14 | #define C2Pconvert(s) do\
 15 |             {int n = strlen(s); memmove(&(s)[1], (s), n); *(s) = n;} while(0)
 16 |
 17 | #if (0)                             /* Demo code fragment follows */
 18 |
 19 |       char string[81];
 20 |
 21 |       fgets(string, 81, inFile);    /* get 80-char C string       */
 22 |       C2Pconvert(string);           /* convert it in place        */
 23 |       P2Cconvert(string);           /* convert back               */
 24 |
 25 | #endif
```

TEXT STATISTICS

```
   731 characters
    25 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     1 constants [character]
     1 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    BYTE            13
    C2Pconvert              14      22
    P2Cconvert              12      23
    do\             14
    fgets           21
    h        9
    inFile          21
    memmove         13      15
    n       13+     15+
    s       12      13+     14      15+
    string           9      19      21      22      23
    strlen          15
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #include<stdio.h>
  4 | #include<stdlib.h>
  5 | #include<string.h>
  6 |
  7 | /*  chouse_n ( char *strng, int length) returns a pointer to a string of   */
  8 | /*  length characters chosen from "strng" , duplicate chars in "strng" are */
  9 | /*  significant.  Strings are generated in lexical order.                   */
 10 | /*  First call, call with *strng. each subsequent call, call with NULL,     */
 11 | /*  returns one combination. Calls after all combinations have been         */
 12 | /*  returned return NULL.  Will return NULL for errors.                     */
 13 | /*  not very defensive (i.e. WILL BREAK)                                    */
 14 |
 15 | /*  dave chapman aug '91  released to public domain                         */
 16 |
 17 | char *chouse_n( char *strng, int length);
 18 |
 19 | char *chouse_n( char *strng, int length)
 20 | {
 21 |       static char *str;
 22 |       static char *curr;
 23 |       static char *pos;       /* for each char in curr(ent string),
 24 |                                  its pos in str                          */
 25 |       static int  counts[256];
 26 |       int i,j;
 27 |
 28 |       if (0 >= length)
 29 |             return NULL;
 30 |
 31 |       if (NULL != strng)
 32 |       {
 33 |             str = malloc(strlen(strng)); /* first call, prep string for use */
 34 |             curr = malloc(2 * length + 1);
 35 |             pos = curr + length +1;
 36 |
 37 |             for (i = 0; i < 256; counts[i++] = 0)
 38 |                   ;
 39 |             for (i = 0; strng[i]; i++)
 40 |                   counts[strng[i]]++;
 41 |
 42 |             for (i = 1, j = 0; i < 256; i++)
 43 |             {
 44 |                   if (counts[i])
 45 |                   {
 46 |                         str[j] = i;
 47 |                         counts[j++] = counts[i];
 48 |                   }
 49 |             }
 50 |             str[j] = '\0';      /* str is string of distinct chars in order */
 51 |                                 /* counts[] holds count of each char        */
 52 |
 53 |             /* take first length chars */
 54 |
 55 |             for (i = 0,j = 0; i < length; i++)
 56 |             {
 57 |                   curr[i] = str[j];
 58 |                   pos[i] = j;
 59 |                   if (!(--counts[j]))
 60 |                         j++;
 61 |             }
 62 |             curr[i] = '\0';
 63 |             return curr;
 64 |       }
 65 |       /* if called with "mississippi",5;
 66 |          str -> "imps"
 67 |          curr -> "iiiim"
 68 |          counts -> 0,0,2,4;
 69 |          pos -> 0,0,0,0,1;   */
 70 |
 71 |       /* go back to front */
 72 |
 73 |       for (j = length; j > 0;)
 74 |       {
 75 |             counts[ pos[--j]]++;                     /* "replace" char */
 76 |
 77 |             /* look for a new char for curr posit. */
 78 |
 79 |             for ( i = ++pos[j]; str[i] && ! counts[i]; i++)
 80 |                   ;
 81 |             if (0 != (curr[j] = str[i]))             /* found a char   */
 82 |             {
 83 |                   --counts[i];
 84 |                   pos[j] = i;
 85 |
 86 |                   /* placed char, fill out rest of string  */
 87 |
 88 |                   for (++j, i = 0; j < length; j++)
 89 |                   {
 90 |                         for ( ; !counts[i]; i++)
 91 |                               ;
 92 |                         curr[j] = str[i];       /* first available char */
 93 |                         --counts[i];
 94 |                         pos[j] = i;
 95 |                   }
 96 |                   return curr;
 97 |             }
 98 |             /* no more chars for this pos ; go back one */
 99 |       }
100 |       /*  done */
```

```
 101 |       return NULL;
 102 | }
 103 |
 104 | main()
 105 | {
 106 |       char  *str = "aabbccdd";
 107 |       int i,j;
 108 |
 109 |       j = 0;
 110 |       i =  5;
 111 |       puts(chouse_n( str, i));
 112 |       while (NULL != (str = chouse_n(NULL, i)))
 113 |       {
 114 |             ++j;
 115 |             printf(" %s  %d\n",str,j);
 116 |       }
 117 |       return 0;
 118 | }
```

TEXT STATISTICS

```
  3637 characters
   118 lines
```

LEXICAL STATISTICS

```
    23 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     2 constants [character]
     2 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    NULL            29      31     101     112+
    chouse_n        17      19     111     112
    counts          25      37      40      44      47+     59      75      79      83      90      93
    curr            22      34      35      57      62      63      81      92      96
    h        3       4       5
    i       26      37+     39+     40      42+     44      46      47      55+     57      58      62
            79+     81      83      84      88      90+     92      93      94     107     110     111
           112
    j       26      42      46      47      50      55      57      58      59      60      73+     75
            79      81      84      88+     92      94     107     109     114     115
    length          17      19      28      34      35      55      73      88
    main           104
    malloc          33      34
    pos             23      35      58      75      79      84      94
    printf         115
    puts           111
    stdio            3
    stdlib           4
    str             21      33      46      50      57      79      81      92     106     111     112
           115
    string           5
    strlen          33
    strng           17      19      31      33      39      40
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PERMUTE.C - prints all permutations of an input string
**
**  public domain demo by Jon Guthrie
*/

#include    <string.h>
#include    <stdlib.h>
#include    <stdio.h>

int     charcmp(char *, char *);

void    permute(char *, int, int);

int     main(int argc, char *argv[])
{
      int length;

      if (2 > argc)
      {
            puts("Usage: PERMUTE string");
            abort();
      }

      length = strlen(argv[1]);

      /* It only works if they're printed in order */

      qsort(argv[1], length, 1, (int(*)(const void *, const void *))charcmp);

      permute(argv[1], 0, length);
      return 0;
}


/*
**  This function prints all of the permutations of string "array"
**  (which has length "len") starting at "start."
*/

void    permute(char *array, int start, int len)
{
      int j;
      char    *s;

      if(start < len)
      {
            if(NULL == (s = malloc(len)))
            {
                  printf("\n\nMemory error!!!\a\a\n");
                  abort();
            }

            strcpy(s, array);
            for(j=start ; j<len ; ++j)
            {
                  int     temp;

                  if((j == start) || (s[j] != s[start]))
                  {     /* For each character that's different    */
                        /* Swap the next first character with...  */
                        /* the current first                      */
                        temp = s[j];
                        s[j] = s[start];
                        s[start] = temp;
                        permute(s, start+1, len);
                  }
            }
            free(s);
      }
      else  puts(array);
}


int charcmp(char *a, char *b)
{
      return(*a - *b);
}
```

TEXT STATISTICS

```
  1845 characters
    81 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     2 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

NULL		50							
a	78	80							
abort		24	53						
argc		17	21						
argv		17	27	31	33				
array		43	56	73					
b	78	80							
charcmp		13	31	78					
free		71							
h	9	10	11						
j	45	57+	61+	65	66				
len		43	48	50	57	68			
length		19	27	31	33				
main		17							
malloc		50							
permute		15	33	43	68				
printf		52							
puts		23	73						
qsort		31							
s	46	50	56	61+	65	66+	67	68	71
start		43	48	57	61+	66	67	68	
stdio		11							
stdlib		10							
strcpy		56							
string		9							
strlen		27							
temp		59	65	67					

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** Determine the permutation index for a given permutation list.
  5 | ** Written by Thad Smith III, Boulder, CO  8/31/91
  6 | ** Hereby contributed to the Public Domain.
  7 | **
  8 | ** The following function computes the ordinal of the given permutation,
  9 | ** which is index of the permutation in sorting order:
 10 | **  1, 2, ..., n-1, n   is index 0
 11 | **  1, 2, ..., n, n-1   is index 1
 12 | **  ...
 13 | **  n, n-1, ..., 2, 1   is index n! -1
 14 | **
 15 | ** The actual values of the elements are immaterial, only the relative
 16 | ** ordering of the values is used.
 17 | **
 18 | ** pit[] is the array of elements of length size.
 19 | ** The return value is the permutation index.
 20 | */
 21 |
 22 | #include "snipmath.h"
 23 |
 24 | int perm_index (char pit[], int size)
 25 | {
 26 |       int i;
 27 |       register int j, ball;
 28 |       int index = 0;
 29 |
 30 |       for (i = 1; i < size; i++)
 31 |       {
 32 |             ball = pit[i-1];
 33 |             for (j = i; j < size; j++)
 34 |             {
 35 |                   if (ball > pit[j])
 36 |                         index ++;
 37 |             }
 38 |             index *= size - i;
 39 |       }
 40 |       return index;
 41 | }
```

TEXT STATISTICS

```
  1101 characters
    41 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    ball            27      32      35
    i       26      30+     32      33      38
    index           28      36      38      40
    j       27      33+     35
    perm_index              24
    pit             24      32      35
    size            24      30      33      38
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  Written and released to the public domain by David Engel.
 *
 *  This function attempts to open a file which may be in any of
 *  several directories.  It is particularly useful for opening
 *  configuration files.  For example, PROG.EXE can easily open
 *  PROG.CFG (which is kept in the same directory) by executing:
 *
 *      cfg_file = pfopen("PROG.CFG", "r", getenv("PATH"));
 *
 *  NULL is returned if the file can't be opened.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "snpdosys.h"

FILE *pfopen(const char *name, const char *mode, const char *dirs)
{
    char *ptr;
    char *tdirs;
    FILE *file = NULL;

    if (dirs == NULL || dirs[0] == '\0')
        return NULL;

    if ((tdirs = malloc(strlen(dirs)+1)) == NULL)
        return NULL;

    strcpy(tdirs, dirs);

    for (ptr = strtok(tdirs, SEP_CHARS); file == NULL && ptr != NULL;
        ptr = strtok(NULL, SEP_CHARS))
    {
        size_t len;
        char work[FILENAME_MAX];

        strcpy(work, ptr);
        len = strlen(work);
        if (len && work[len-1] != '/' && work[len-1] != '\\')
            strcat(work, "/");
        strcat(work, name);

        file = fopen(work, mode);
    }

    free(tdirs);

    return file;
}

#ifdef TEST

int main(int argc, char **argv)
{
    FILE *file;

    if (argc != 4)
    {
        fprintf(stderr, "usage: pfopen name mode dirs\n");
        exit(1);
    }

    file = pfopen(argv[1], argv[2], argv[3]);

    printf("%s \"%s\" with mode \"%s\"\n", (file == NULL) ?
        "Could not open" : "Opened", argv[1], argv[2]);

    return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1728 characters
    75 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     3 constants [character]
     6 constants [string]
    12 constants [numeric]
```

SYMBOL TABLE

FILE	21	25	59					
FILENAME_MAX		39						
NULL	25	27	28	30	31	35+	36	69
SEP_CHARS	35	36						
TEST	55							
argc	57	61						
argv	57	67+	70+					
dirs	21	27+	30	33				
exit	64							
file	25	35	47	52	59	67	69	
fopen	47							
fprintf	63							
free	50							
h 16	17	18						
len	38	42	43+					
main	57							
malloc	30							
mode	21	47						
name	21	45						
pfopen	21	67						
printf	69							
ptr	23	35+	36	41				
size_t	38							
stderr	63							
stdio	16							
stdlib	17							
strcat	44	45						
strcpy	33	41						
string	18							
strlen	30	42						
strtok	35	36						
tdirs	24	30	33	35	50			
work	39	41	42	43+	44	45	47	

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PHONETIC.H - Snippets header file for functions to perform
**               phonetic string matching.
*/

#ifndef PHONETIC__H
#define PHONETIC__H

#include "sniptype.h"


/*
**  File SOUNDEX.C
*/

char *soundex(char *instr, char *outstr);


/*
**  File SOUNDEX4.C
*/

void  soundex4(const char *instr, char *outstr, int N);


/*
**  File SOUNDEX5.C
*/

#define SNDMAX (1 + 6 + 6*6 + 6*6*6)
#define SNDLEN (26 * SNDMAX)

int soundex5(char *instr);


/*
**  File METAPHON.C
*/

/*
**  MAXMETAPH is the length of the Metaphone code.
**
**  Four is a good compromise value for English names. For comparing words
**  which are not names or for some non-English names, use a longer code
**  length for more precise matches.
**
**  The default here is 5.
*/

#define MAXMETAPH 5

typedef enum {COMPARE, GENERATE} metaphlag;

Boolean_T metaphone(const char *Word, char *Metaph, metaphlag Flag);


/*
**  File APPROX.C
*/

void App_init(char *pattern, char *text, int degree);
void App_next(char **start, char **end, int *howclose);

#endif /* PHONETIC__H */
```

```
TEXT STATISTICS

  1163 characters
    66 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     1 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    App_init        63
    App_next        64
    Boolean_T       56
    COMPARE         54
    Flag            56
    GENERATE        54
    MAXMETAPH       52
    Metaph          56
    N       25
    PHONETIC__H              8       9
    SNDLEN          33
    SNDMAX          32      33
    Word            56
    degree          63
    end             64
    howclose        64
    instr           18      25      35
    metaphlag       54      56
    metaphone       56
    outstr          18      25
    pattern         63
    soundex         18
    soundex4        25
    soundex5        35
    start           64
    text            63
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PI.C - Computes Pi to an arbitrary number of digits
**
**  Uses far arrays when/where required so may be compiled in any memory model
**
**  The formula that most use (including the one in the Snippets) is the
**  classic Machin formula, with the Gregory series.
**
**  pi=16arctan(1/5)-4arctan(1/239)
**
**  And use the traditional Gregory arctangent series to calculate the
**  arctangents. That's the:
**
**  arctan(x)=x-(x^3)/3+(x^5)/5-(x^7)/7.....
**
**  With 1/5 and 1/239, it would be:
**
**  arctan(x)=1/5-1/(3*5^3)+1/(5*5^5)-1/(7*5^7)...
**  arctan(x)=1/239-1/(3*239^3)+1/(5*239^5)-1/(7*239^7)....
**
**  Doing the multi-precision isn't too hard, since we don't really need to
**  have a general purpose math package.  We can hardwire it all.
**
**  Due to the % operator, ms[i] < (temp * (239**2)) so
**  temp < 3759 and i < 1879, it fails at the 1879th term which translates to
**  1879 * log10(239**2) == 8941th decimal.
**
**  In practice we get a few more digits, (2 -> 8943th)
*/

#include <stdio.h>
#include <stdlib.h>
#include "big_mall.h"

long kf, ks;
long FAR *mf, FAR *ms;
long cnt, n, temp, nd;
long i, line;
long col, col1;
long loc, FAR stor[4096];

static void PASCAL shift(long FAR *l1, long FAR *l2, long lp, long lmod)
{
      long k;

      k = ((*l2) > 0 ? (*l2) / lmod: -(-(*l2) / lmod) - 1);
      *l2 -= k * lmod;
      *l1 += k * lp;
}

static void PASCAL yprint(long m)
{
      if (cnt<n)
      {
            if (++col == 11)
            {
                  col = 1;
                  if (++col1 == 5)
                  {
                        col1 = 0;
                        printf("\nL%04ld:", ++line);
                        printf("%4ld",m%10);
                  }
                  else  printf("%3ld",m%10);
            }
            else  printf("%ld",m);
            cnt++;
      }
}

static void PASCAL xprint(long m)
{
      long ii, wk, wk1;

      if (m < 8)
      {
            for (ii = 1; ii <= loc; )
                  yprint(stor[(int)(ii++)]);
            loc = 0;
      }
      else
      {
            if (m > 9)
            {
                  wk = m / 10;
                  m %= 10;
                  for (wk1 = loc; wk1 >= 1; wk1--)
                  {
                        wk += stor[(int)wk1];
                        stor[(int)wk1] = wk % 10;
                        wk /= 10;
                  }
            }
      }
      stor[(int)(++loc)] = m;
}

static void PASCAL memerr(int errno)
```

```
101 | {
102 |         printf("\a\nOut of memory error #%d\n", errno);
103 |         if (2 == errno)
104 |                 BigFree(mf);
105 |         _exit(2);
106 | }
107 |
108 | int main(int argc, char *argv[])
109 | {
110 |       int i=0;
111 |       char *endp;
112 |
113 |       stor[i++] = 0;
114 |       if (argc < 2)
115 |       {
116 |             puts("\aUsage: PI <number_of_digits>");
117 |             return(1);
118 |       }
119 |       n = strtol(argv[1], &endp, 10);
120 |       mf = BigMalloc((Size_T)(n + 3L), (Size_T)sizeof(long));
121 |       if (NULL == mf)
122 |             memerr(1);
123 |       ms = BigMalloc((Size_T)(n + 3L), (Size_T)sizeof(long));
124 |       if (NULL == ms)
125 |             memerr(2);
126 |       printf("\nApproximation of PI to %ld digits\n", (long)n);
127 |       cnt = 0;
128 |       kf = 25;
129 |       ks = 57121L;
130 |       mf[1] = 1L;
131 |       for (i = 2; i <= (int)n; i += 2)
132 |       {
133 |             mf[i] = -16L;
134 |             mf[i+1] = 16L;
135 |       }
136 |       for (i = 1; i <= (int)n; i += 2)
137 |       {
138 |             ms[i] = -4L;
139 |             ms[i+1] = 4L;
140 |       }
141 |       printf("\nL%04ld: 3.", ++line);
142 |       while (cnt < n)
143 |       {
144 |             for (i = 0; ++i <= (int)n - (int)cnt; )
145 |             {
146 |                   mf[i] *= 10L;
147 |                   ms[i] *= 10L;
148 |             }
149 |             for (i =(int)(n - cnt + 1); --i >= 2; )
150 |             {
151 |                   temp = 2 * i - 1;
152 |                   shift(&mf[i - 1], &mf[i], temp - 2, temp * kf);
153 |                   shift(&ms[i - 1], &ms[i], temp - 2, temp * ks);
154 |             }
155 |             nd = 0;
156 |             shift((long FAR *)&nd, &mf[1], 1L, 5L);
157 |             shift((long FAR *)&nd, &ms[1], 1L, 239L);
158 |             xprint(nd);
159 |       }
160 |       printf("\n\nCalculations Completed!\n");
161 |       BigFree(ms);
162 |       BigFree(mf);
163 |       return(0);
164 | }
```

TEXT STATISTICS

```
  4335 characters
   164 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    10 constants [string]
    64 constants [numeric]
```

SYMBOL TABLE

```
    BigFree        104     161     162
    BigMalloc      120     123
    FAR             38+     42      44+    156     157
    NULL           121     124
    PASCAL          44      53      73     100
    Size_T         120+    123+
    _exit          105
    argc           108     114
    argv           108     119
    cnt             39      55      69     127     142     144     149
    col             41      57      59
    col1            41      60      62
    endp           111     119
    errno          100     102     103
    h       33      34
    i       40     110     113     131+    133     134     136+    138     139     144+    146     147
           149+    151     152+    153+
    ii              75      79+     80
    k       46      48      49      50
    kf              37     128     152
    ks              37     129     153
    l1              44      50
    l2              44      48+     49
    line            40      63     141
    lmod            44      48+     49
    loc             42      79      81      89      97
    lp              44      50
    m       53      64      66      68      73      77      85      87      88      97
    main           108
    memerr         100     122     125
    mf              38     104     120     121     130     133     134     146     152+    156     162
    ms              38     123     124     138     139     147     153+    157     161
    n       39      55     119     120     123     126     131     136     142     144     149
    nd              39     155     156     157     158
    printf          63      64      66      68     102     126     141     160
    puts           116
    shift           44     152     153     156     157
    stdio           33
    stdlib          34
    stor            42      80      91      92      97     113
    strtol         119
    temp            39     151     152+    153+
    wk              75      87      91      92      93
    wk1             75      89+     91      92
    xprint          73     158
    yprint          53      80
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | #ifndef PI__H
   4 | #define PI__H
   5 |
   6 | #ifndef PI
   7 |  #define PI         (4*atan(1))
   8 | #endif
   9 |
  10 | #define deg2rad(d) ((d)*PI/180)
  11 | #define rad2deg(r) ((r)*180/PI)
  12 |
  13 | #endif /* PI__H */
```

TEXT STATISTICS

```
   219 characters
    13 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     0 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    PI               6       7      10      11
    PI__H            3       4
    atan             7
    d       10+
    deg2rad         10
    r       11+
    rad2deg         11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
  pi8.c  Sept 9, 1996.

  This program is placed into the public domain by its author,
  Carey Bloodworth.

  This pi program can calculate pi with either integers, or doubles.
  It can also use a variety of formulas.

  This code isn't really optimized because it has to work with both the
  long integer version and the double version, and several formulas.
  Compromises had to be made in several places.

  When compiling, you can chose to use the FPU or the integer version.
  By default, it will chose to work only in integers.  If you want to
  use the FPU, define:

#define USEFPU     1

  You have a choice of formulas.  By default, it will use the Machin
  formula of:  pi=16arctan(1/5)-4arctan(1/239)

  You could chose to use one of the other formulas.

  for pi=8arctan(1/3)+4arctan(1/7)
#define ARC3  1
  for pi=12arctan(1/4)+4arctan(1/20)+4arctan(1/1985)
#define ARC4  1
  for pi=16arctan(1/5)-4arctan(1/70)+4arctan(1/99)
#define ARC5  1
  for pi=32arctan(1/10)-4arctan(1/239)-16arctan(1/515)
#define ARC10 1

  Or, of course, you could define it on the compile command line with
  the -D switch.

  Timings were done on a Cyrix 486/66, with the slow Turbo C++ v3.0
          1,000 2,000 3,000 4,000  1,000F 2,000F 3,000F 4,000F
  Machin      4    18    45    86      1      5     11     20
  Arc3        6    29    74   140      2      9     20     35
  Arc4        5    24    64   116      2      7     16     29
  Arc5        6    26    65   123      1      6     15     26
  Arc10       4    19    46    86      1      5     11     19

  All of the combinations above were verified to their indicated
  precision and in each case, only the last few digits were wrong,
  which is a normal round off / truncation error.

  Better compilers will of course result in faster computations,
  but the ratios should be the same.  When I used GCC 2.6.3 for
  DOS, I computed 4,000 digits with with the Machin formula and
  the FPU in 8 seconds.  The integer version took 17 seconds.

  I also used the FPU GCC version to compute 65,536 digits of
  pi and verified them against the Gutenberg PIMIL10.TXT, and
  only the last 4 digits were incorrect.  The computations took
  33 minutes and 54 seconds.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SHOWTIME
#define USEFPU

#if defined USEFPU

#define BASE   1000000000L
#define BASEDIGITS 9

typedef long int SHORT;
typedef double   LONG;
#else

#define BASE   100
#define BASEDIGITS 2

typedef unsigned char SHORT;
typedef long int      LONG;
#endif

typedef long int INDEXER;


SHORT *pi, *powers, *term;
INDEXER size;


void OutDig(int dig)
{
      static int printed = 0;

      putchar(dig + '0');
      printed++;
      if ((printed%1000) == 0)
      {
            printed = 0;
```

```
101 |             printf("\n\n\n");
102 |       }
103 |       if ((printed%50) == 0)
104 |             printf("\n");
105 |       else if ((printed%10) == 0)
106 |             putchar(' ');
107 | }
108 |
109 |
110 | void PrintShort(SHORT num)
111 | {
112 |       int x;
113 |       int digits[BASEDIGITS + 1];
114 |
115 |       for (x = 0; x < BASEDIGITS; x++)
116 |       {
117 |             digits[x] = num % 10;
118 |             num /= 10;
119 |       }
120 |       for (x = BASEDIGITS - 1; x >= 0; x--)
121 |             OutDig(digits[x]);
122 | }
123 |
124 | void Print(SHORT *num)
125 | {
126 |       INDEXER x;
127 |
128 |       printf("\nPI = 3.\n");
129 |       for (x = 1; x < size; x++)
130 |             PrintShort(num[x]);
131 |       printf("\n");
132 | }
133 |
134 | void arctan(int multiplier, int denom, int sign)
135 | {
136 |       INDEXER x;
137 |       LONG remain, temp, divisor, denom2;
138 |       SHORT NotZero = 1;
139 |       INDEXER adv;
140 |
141 |       for (x = 0; x < size; x++)
142 |             powers[x] = 0;
143 |
144 |       divisor = 1;
145 |       denom2 = (LONG)denom;denom2 *= denom2;
146 |       adv = 0;
147 |
148 |       remain = (LONG)multiplier * denom;
149 |       while (NotZero)
150 |       {
151 |             for (x = adv; x < size; x++)
152 |             {
153 |                   temp = (LONG)powers[x] + remain;
154 |                   powers[x] = (SHORT)(temp / denom2);
155 |                   remain = (temp - (denom2 * (LONG)powers[x])) * BASE;
156 |             }
157 |
158 |             remain = 0;
159 |             for (x = adv; x < size; x++)
160 |             {
161 |                   temp = (LONG)powers[x] + remain;
162 |                   term[x] = (SHORT)(temp / divisor);
163 |                   remain = (temp - (divisor * (LONG)term[x])) * BASE;
164 |             }
165 |             remain = 0;
166 |
167 |             if (sign > 0)
168 |             {
169 |                   LONG carry, sum;
170 |
171 |                   carry = 0;
172 |                   for (x = size - 1; x >=0; x--)
173 |                   {
174 |                         sum = (LONG)pi[x] + (LONG)term[x] + carry;
175 |                         carry = 0;
176 |                         if (sum >= BASE)
177 |                         {
178 |                               carry = 1;
179 |                               sum -= BASE;
180 |                         }
181 |                         pi[x] = (SHORT)sum;
182 |                   }
183 |             }
184 |             else
185 |             {
186 |                   LONG borrow, sum;
187 |
188 |                   borrow = 0;
189 |                   for (x = size - 1; x >= 0; x--)
190 |                   {
191 |                         sum = (LONG)pi[x] - (LONG)term[x] - borrow;
192 |                         borrow = 0;
193 |                         if (sum < 0)
194 |                         {
195 |                               borrow = 1;
196 |                               sum += BASE;
197 |                         }
198 |                         pi[x] = (SHORT)sum;
199 |                   }
200 |             }
```

```
201 |
202 |             sign = -sign;
203 |             divisor += 2;
204 |             NotZero = 0;
205 |             for (x = adv; x < size; x++)
206 |             {
207 |                   if (powers[x])
208 |                   {
209 |                         NotZero = 1;
210 |                         break;
211 |                   }
212 |             }
213 |
214 |             if (NotZero)
215 |             {
216 |                   while (powers[adv] == 0)
217 |                         adv++;
218 |             }
219 |             /* We can skip ones that are already 0 */
220 |       }
221 | }
222 |
223 | int main(int argc, char *argv[])
224 | {
225 |       INDEXER x;
226 |       time_t T1, T2;
227 |
228 |       if (argc != 2)
229 |       {
230 |             printf("I need to know how many digits to compute.\n");
231 |             exit(EXIT_FAILURE);
232 |       }
233 |
234 |       size = atol(argv[1]);
235 |       if (size <= 0L)
236 |       {
237 |             printf("Invalid argument.\n");
238 |             exit(EXIT_FAILURE);
239 |       }
240 |
241 |       size = ((size + BASEDIGITS - 1) / BASEDIGITS) + 1;
242 |
243 |       pi = malloc(sizeof(SHORT) * size);
244 |       powers = malloc(sizeof(SHORT) * size);
245 |       term = malloc(sizeof(SHORT) * size);
246 |
247 |       if ((pi == NULL) || (powers == NULL) || (term == NULL))
248 |       {
249 |             printf("Unable to allocate enough memory.\n");
250 |             exit(EXIT_FAILURE);
251 |       }
252 |
253 |       for (x = 0; x < size; x++)
254 |             pi[x] = 0;
255 |
256 |       T1 = time(NULL);
257 |
258 | #if defined ARC3
259 |       arctan(8, 3, 1);
260 |       arctan(4, 7, 1);
261 | #elif defined ARC5
262 |       arctan(16, 5, 1);
263 |       arctan(4, 70, -1);
264 |       arctan(4, 99, 1);
265 | #elif defined  ARC4
266 |       arctan(12, 4, 1);
267 |       arctan(4, 20, 1);
268 |       arctan(4, 1985, 1);
269 | #elif defined  ARC10
270 |       arctan(32, 10, 1);
271 |       arctan(4, 239, -1);
272 |       arctan(16, 515, -1);
273 | #else
274 |       /* Machin formula */
275 |       arctan(16, 5, 1);
276 |       arctan(4, 239, -1);
277 | #endif
278 |
279 |       T2 = time(NULL);
280 |
281 |       Print(pi);
282 |
283 | #ifdef SHOWTIME
284 |       printf("\nCalculation time %0.0lf\n", difftime(T2, T1));
285 | #endif
286 |
287 |       return EXIT_SUCCESS;
288 | }
```

TEXT STATISTICS

```
  7515 characters
   288 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
     2 constants [character]
     8 constants [string]
    88 constants [numeric]
```

SYMBOL TABLE

```
    ARC10           269
    ARC3            258
    ARC4            265
    ARC5            261
    BASE             71      78     155     163     176     179     196
    BASEDIGITS               72      79     113     115     120     241+
    EXIT_FAILURE            231     238     250
    EXIT_SUCCESS            287
    INDEXER          85      89     126     136     139     225
    LONG             75      82     137     145     148     153     155     161     163     169     174+
           186      191+
    NULL            247+    256     279
    NotZero         138     149     204     209     214
    OutDig           92     121
    Print           124     281
    PrintShort              110     130
    SHORT            74      81      88     110     124     138     154     162     181     198     243
           244      245
    SHOWTIME         66     283
    T1              226     256     284
    T2              226     279     284
    USEFPU           67      69
    adv             139     146     151     159     205     216     217
    arctan          134     259     260     262     263     264     266     267     268     270     271
           272      275     276
    argc            223     228
    argv            223     234
    atol            234
    borrow          186     188     191     192     195
    carry           169     171     174     175     178
    defined          69     258     261     265     269
    denom           134     145     148
    denom2          137    145+     154     155
    difftime        284
    dig              92      96
    digits          113     117     121
    divisor         137     144     162     163     203
    exit            231     238     250
    h        62      63      64
    main            223
    malloc          243     244     245
    multiplier              134     148
    num             110     117     118     124     130
    pi               88     174     181     191     198     243     247     254     281
    powers           88     142     153     154     155     161     207     216     244     247
    printed          94      97      98     100     103     105
    printf          101     104     128     131     230     237     249     284
    putchar          96     106
    remain          137     148     153     155     158     161     163     165
    sign            134     167    202+
    size             89     129     141     151     159     172     189     205     234     235     241+
           243      244     245     253
    stdio            62
    stdlib           63
    sum             169     174     176     179     181     186     191     193     196     198
    temp            137     153     154     155     161     162     163
    term             88     162     163     174     191     245     247
    time             64     256     279
    time_t          226
    x       112     115+    117     120+    121     126     129+    130     136     141+    142     151+
            153     154     155     159+    161     162     163     172+    174+    181     189+    191+
            198     205+    207     225     253+    254
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** This method implements the Salamin / Brent / Gauss Arithmetic-
** Geometric Mean pi formula.
**
** Let A[0] = 1, B[0] = 1/Sqrt(2)
**
** Then iterate from 1 to 'n'.
** A[n] = (A[n-1] + B[n-1])/2
** B[n] = Sqrt(A[n-1]*B[n-1])
** C[n]^2 = A[n]^2 - B[n]^2  (or) C[n] = (A[n-1]-B[n-1])/2
**                          n
** PI[n] = 4A[n+1]^2 / (1-(Sum (2^(j+1))*C[j]^2))
**                         j = 1
**
** There is an actual error calculation, but it comes out  to  slightly
** more  than double on each iteration.  I think it results in about 17
** million  correct  digits,  instead  of  16  million  if  it actually
** doubled.  PI16 generates 178,000 digits.  PI19 to  over  a  million.
** PI22 is 10 million, and PI26 to 200 million.
**
** For what little it's worth, this program is placed into the public
** domain by its author, Carey Bloodworth, on September 21, 1996.
**
** The math routines in this program are not general purpose routines.
** They have been optimized and written for this specific use.  The
** concepts are of course portable, but not the implementation.
**
** The program run time is about O(n^1.7).  That's fairly good growth,
** compared to things such as the classic arctangents.  But not as good
** as it should be, if I used a FFT based multiplication.  Also, the 'O'
** is fairly large.  In fact, I'd guess that this program could compute
** one million digits of pi in about the same time as my previously
** posted arctangent method, in spite of this one having less than n^2
** growth.
**
** The program has not been cleaned up.  It's written rather crudely
** and dirty.  The RSqrt() in particular is rather dirty, having gone
** through numerous changes and attempts at speeding it up.
** But I'm not planning on doing any more with it.  The growth isn't as
** low as I'd hoped, and until I find a faster multiplication, the
** method isn't any better than simpler arctangents.
**
** I currently use a base of 10,000 simply because it made debugging
** easier.  A base of 65,536 and modifying the FastMul() to handle sizes
** that aren't powers of two would allow a bit better efficiency.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "snipmath.h"

#ifdef __TURBOC__
typedef short int Indexer;
#else
typedef long int Indexer;
#endif

typedef short int Short;
typedef long  int Long;

Short *a, *b, *c, *Work1, *MulWork, *Work2, *Work3;
int SIZE;

/*
** Work1 is explicitly used in Reciprocal() and RSqrt()
** Work1 is implicitly used in Divide() and Sqrt()
** Work2 is explicitly used in Divide() and Sqrt()
** Work3 is used only in the AGM and holds the previous reciprocal
**  square root, allowing us to save some time in RSqrt()
*/


void Copy(Short *a, Short *b, Indexer Len)
{
      while (Len--) a[Len] = b[Len];
}

/*
** This rounds and copies a 2x mul result into a normal result
** Our number format will never have more than one unit of integer,
** and after a mul, we have two, so we need to fix that.
*/

void Round2x(Short *a, Short *b, Indexer Len)
{
      Indexer x;
      Short carry;

      carry = 0;
      if (b[Len+1] >= 5000)
            carry = 1;
      for (x = Len; x > 0; x--)
      {
            carry += b[x];
            a[x-1] = carry % 10000;
            carry /= 10000;
```

```
101 |       }
102 | }
103 |
104 | void DivBy2(Short *n, Indexer Len)
105 | {
106 |       Indexer x;
107 |       Long temp;
108 |
109 |       temp = 0;
110 |       for (x = 0; x < Len; x++)
111 |       {
112 |             temp = (Long)n[x]+temp*10000;
113 |             n[x] = (Short)(temp/2);
114 |             temp = temp%2;
115 |       }
116 | }
117 |
118 | void DoCarries(Short *limit, Short *cur, Short carry)
119 | {
120 |       Long temp;
121 |
122 |       while ((cur >= limit) && (carry != 0))
123 |       {
124 |             temp = *cur+carry;
125 |             carry = 0;
126 |             if (temp >= 10000)
127 |             {
128 |                   carry = 1;
129 |                   temp -= 10000;
130 |             }
131 |             *cur = temp;
132 |             --cur;
133 |       }
134 | }
135 |
136 | void DoBorrows(Short *limit, Short *cur, Short borrow)
137 | {
138 |       Long temp;
139 |       while ((cur >= limit) && (borrow != 0))
140 |       {
141 |             temp = *cur-borrow;
142 |             borrow = 0;
143 |             if (temp < 0)
144 |         {
145 |             borrow = 1;
146 |             temp += 10000;
147 |         }
148 |             *cur = temp;
149 |             --cur;
150 |       };
151 | }
152 |
153 | void PrintShort2(char *str, Short *num, Indexer Len)
154 | {
155 |       Indexer x;
156 |
157 |       printf("%s ", str);
158 |       printf("%u.", num[0]);
159 |       for (x = 1; x < Len; x++)
160 |             printf("%04u", num[x]);
161 |       printf("\n");
162 | }
163 |
164 | void PrintShort(char *str, Short *num, Indexer Len)
165 | {
166 |       Indexer x;
167 |       int printed = 0;
168 |
169 |       printf("%s ", str);
170 |
171 |       printf("%u.\n", num[0]);
172 |
173 |       for (x = 1; x < Len; x++)
174 |       {
175 |             printf("%02d", num[x]/100);
176 |             printed += 2;
177 |             if ((printed % 1000) == 0)
178 |             {
179 |                   printf("\n\n\n\n");
180 |                   printed = 0;
181 |             }
182 |             else if ((printed % 50) == 0)
183 |             {
184 |                   printf("\n");
185 |             }
186 |             else if ((printed % 10) == 0)
187 |             {
188 |                   printf(" ");
189 |             }
190 |             printf("%02d", num[x] % 100);
191 |             printed += 2;
192 |             if ((printed % 1000) == 0)
193 |             {
194 |                   printf("\n\n\n\n");
195 |                   printed = 0;
196 |             }
197 |             else if ((printed % 50) == 0)
198 |             {
199 |                   printf("\n");
200 |             }
```

```
201 |             else if ((printed % 10) == 0)
202 |             {
203 |                   printf(" ");
204 |             }
205 |       }
206 |       printf("\n");
207 |
208 | }
209 |
210 | /* sum = a + b */
211 |
212 | Short Add(Short *sum, Short *a, Short *b, Indexer Len)
213 | {
214 |       Long s, carry;
215 |       Indexer x;
216 |
217 |       carry = 0;
218 |       for (x = Len - 1; x >= 0; x--)
219 |       {
220 |             s = (Long)a[x] + (Long)b[x] + carry;
221 |             sum[x] = (Short)(s%10000);
222 |             carry = s/10000;
223 |       }
224 |       return carry;
225 | }
226 |
227 | /* dif = a-b */
228 |
229 | Short Sub(Short *dif, Short *a, Short *b, Indexer Len)
230 | {
231 |       Long d, borrow;
232 |       Indexer x;
233 |
234 |       borrow = 0;
235 |       for (x = Len - 1; x >= 0; x--)
236 |       {
237 |             d = (Long)a[x] - (Long)b[x] - borrow;
238 |             borrow = 0;
239 |             if (d < 0)
240 |             {
241 |                   borrow = 1;
242 |                   d += 10000;
243 |             }
244 |             dif[x] = (Short)d;
245 |       }
246 |       return borrow;
247 | }
248 |
249 | void Negate(Short *num, Indexer Len)
250 | {
251 |       Indexer x;Long d, borrow;
252 |
253 |       borrow = 0;
254 |       for (x = Len - 1; x >= 0; x--)
255 |       {
256 |             d = 0 - num[x] - borrow;
257 |             borrow = 0;
258 |             if (d < 0)
259 |             {
260 |                   borrow = 1;
261 |                   d += 10000;
262 |             }
263 |             num[x] = (Short)d;
264 |       }
265 | }
266 |
267 | /* prod = a*b.  prod should be twice normal length */
268 |
269 | void Mul(Short *prod, Short *a, Short *b, Indexer Len)
270 | {
271 |       Long p;
272 |       Indexer ia, ib, ip;
273 |
274 |       if ((prod == a) || (b == prod))
275 |       {
276 |             printf("MUL product can't be one of the other arguments\n");
277 |             exit(1);
278 |       }
279 |
280 |       for (ip = 0; ip < Len * 2; ip++)
281 |             prod[ip] = 0;
282 |       for (ib = Len - 1; ib >= 0; ib--)
283 |       {
284 |             if (b[ib] == 0)
285 |                   continue;
286 |             ip = ib + Len;
287 |             p = 0;
288 |             for (ia = Len - 1; ia >= 0; ia--)
289 |             {
290 |                   p = (Long)a[ia]*(Long)b[ib] + p + prod[ip];
291 |                   prod[ip] = p%10000;
292 |                   p = p/10000;
293 |                   ip--;
294 |             }
295 |             while ((p) && (ip >= 0))
296 |             {
297 |                   p += prod[ip];
298 |                   prod[ip] = p%10000;
299 |                   p = p/10000;
300 |                   ip--;
```

```
301 |             }
302 |       }
303 | }
304 |
305 | /*
306 | ** This is based on the simple O(n^1.585) method, although my
307 | ** growth seems to be closer to O(n^1.7)
308 | **
309 | ** It's fairly simple.  a*b is: a2b2(B^2+B)+(a2-a1)(b1-b2)B+a1b1(B+1)
310 | **
311 | ** For a = 4711 and b = 6397, a2 = 47 a1 = 11 b2 = 63 b1 = 97 Base = 100
312 | **
313 | ** If we did that the normal way, we'd do
314 | ** a2b2 = 47*63 = 2961
315 | ** a2b1 = 47*97 = 4559
316 | ** a1b2 = 11*63 =  693
317 | ** a1b1 = 11*97 = 1067
318 | **
319 | ** 29 61
320 | **    45 59
321 | **     6 93
322 | **       10 67
323 | ** -----------
324 | ** 30 13 62 67
325 | **
326 | ** Or, we'd need N*N multiplications.
327 | **
328 | ** With the 'fractal' method, we compute:
329 | ** a2b2 = 47*63 = 2961
330 | ** (a2-b1)(b1-b2) = (47-11)(97-63) = 36*34 = 1224
331 | ** a1b1 = 11*97 = 1067
332 | **
333 | ** 29 61
334 | **    29 61
335 | **    12 24
336 | **    10 67
337 | **       10 67
338 | ** -----------
339 | ** 30 13 62 67
340 | **
341 | ** We need only 3 multiplications, plus a few additions.  And of course,
342 | ** additions are a lot simpler and faster than multiplications, so we
343 | ** end up ahead.
344 | **
345 | ** The way it is written requires that the size to be a power of two.
346 | ** That's why the program itself requires the size to be a power of two.
347 | ** There is no actual requirement in the method, it's just how I did it
348 | ** because I would be working close to powers of two anyway, and it was
349 | ** easier.
350 | */
351 |
352 | void FastMul(Short *prod, Short *a, Short *b, Indexer Len)
353 | {
354 |       Indexer x, HLen;
355 |       int SignA, SignB;
356 |       Short *offset;
357 |       Short *NextProd;
358 |
359 |       /*
360 |       ** This is the threshold where it's faster to do it the ordinary way
361 |       ** On my computer, it's 16.  Yours may be different.
362 |       */
363 |
364 |       if (Len <= 16 )
365 |       {
366 |             Mul(prod, a, b, Len);
367 |             return;
368 |       }
369 |
370 |       HLen = Len/2;
371 |       NextProd = prod + 2*Len;
372 |
373 |       SignA = Sub(prod, a, a + HLen, HLen);
374 |       if (SignA)
375 |       {
376 |             SignA = 1;
377 |             Negate(prod, HLen);
378 |       }
379 |       SignB = Sub(prod + HLen, b + HLen, b, HLen);
380 |       if (SignB)
381 |       {
382 |             SignB = 1;
383 |             Negate(prod + HLen, HLen);
384 |       }
385 |
386 |       FastMul(NextProd, prod, prod + HLen, HLen);
387 |       for (x = 0; x < 2 * Len; x++)
388 |             prod[x] = 0;
389 |       offset = prod + HLen;
390 |       if (SignA == SignB)
391 |             DoCarries(prod, offset - 1, Add(offset, offset, NextProd, Len));
392 |       else  DoBorrows(prod, offset - 1, Sub(offset, offset, NextProd, Len));
393 |
394 |       FastMul(NextProd, a, b, HLen);
395 |       offset = prod + HLen;
396 |       DoCarries(prod, offset - 1, Add(offset, offset, NextProd, Len));
397 |       Add(prod, prod, NextProd, Len);
398 |
399 |       FastMul(NextProd, a + HLen, b + HLen, HLen);
400 |       offset = prod + HLen;
```

```
401 |       DoCarries(prod, offset - 1, Add(offset, offset, NextProd, Len));
402 |       offset = prod + Len;
403 |       DoCarries(prod, offset - 1, Add(offset, offset, NextProd, Len));
404 | }
405 |
406 | /*
407 | ** Compute the reciprocal of 'a'.
408 | ** x[k+1] = x[k]*(2-a*x[k])
409 | */
410 |
411 | void Reciprocal(Short *r, Short *n, Indexer Len)
412 | {
413 |       Indexer x, SubLen;
414 |       int iter;
415 |       double t;
416 |
417 |       /* Estimate our first reciprocal */
418 |
419 |       for (x = 0; x < Len; x++)
420 |             r[x] = 0;
421 |       t = n[0] + n[1]*1.0e-4 + n[2]*1.0e-8;
422 |       if (t == 0.0)
423 |             t = 1;
424 |       t = 1.0/t;
425 |       r[0] = t;
426 |       t = (t - floor(t))*10000.0;
427 |       r[1] = t;
428 |       t = (t - floor(t))*10000.0;
429 |       r[2] = t;
430 |
431 |       iter = log(SIZE)/log(2.0) + 1;
432 |       SubLen = 1;
433 |       while (iter--)
434 |       {
435 |             SubLen *= 2;
436 |             if (SubLen > SIZE)
437 |                   SubLen = SIZE;
438 |             FastMul(MulWork, n, r, SubLen);
439 |             Round2x(Work1, MulWork, SubLen);
440 |
441 |             MulWork[0] = 2;
442 |             for (x = 1; x < Len; x++)
443 |                   MulWork[x] = 0;
444 |             Sub(Work1, MulWork, Work1, SubLen);
445 |
446 |             FastMul(MulWork, r, Work1, SubLen);
447 |             Round2x(r, MulWork, SubLen);
448 |       }
449 | }
450 |
451 | /*
452 | ** Computes the reciprocal of the square root of 'a'
453 | ** y[k+1] = y[k]*(3-a*y[k]^2)/2
454 | **
455 | **
456 | ** We can save quite a bit of time by using part of our previous
457 | ** results!  Since the number is converging onto a specific value,
458 | ** at least part of our previous result will be correct, so we
459 | ** can skip a bit of work.
460 | */
461 |
462 | void RSqrt(Short *r, Short *n, Indexer Len, Indexer SubLen)
463 | {
464 |       Indexer x;
465 |       int iter;
466 |       double t;
467 |
468 |       /* Estimate our first reciprocal square root */
469 |
470 |       if (SubLen < 2 )
471 |       {
472 |             for (x = 0; x < Len; x++)
473 |                   r[x] = 0;
474 |             t = n[0] + n[1]*1.0e-4 + n[2]*1.0e-8;
475 |             if (t == 0.0)
476 |                   t = 1;
477 |             t = 1.0/sqrt(t);
478 |             r[0] = t;
479 |             t = (t - floor(t))*10000.0;
480 |             r[1] = t;
481 |             t = (t - floor(t))*10000.0;
482 |             r[2] = t;
483 |       }
484 |
485 |       /*
486 |       ** PrintShort2("\n  ~R: ", r, SIZE);
487 |       */
488 |
489 |       if (SubLen > SIZE)
490 |             SubLen = SIZE;
491 |       iter = SubLen;
492 |       while (iter <= SIZE*2)
493 |       {
494 |             FastMul(MulWork, r, r, SubLen);
495 |             Round2x(Work1, MulWork, SubLen);
496 |             FastMul(MulWork, n, Work1, SubLen);
497 |             Round2x(Work1, MulWork, SubLen);
498 |
499 |             MulWork[0] = 3;
500 |             for (x = 1; x < SubLen; x++)
```

```
501 |                   MulWork[x] = 0;
502 |             Sub(Work1, MulWork, Work1, SubLen);
503 |
504 |             FastMul(MulWork, r, Work1, SubLen);
505 |             Round2x(r, MulWork, SubLen);
506 |             DivBy2(r, SubLen);
507 |
508 |             /*
509 |             printf("%3d", SubLen);
510 |             PrintShort2("R: ", r, SubLen);
511 |             printf("%3d", SubLen);
512 |             PrintShort2("R: ", r, Len);
513 |             */
514 |
515 |             SubLen *= 2;
516 |             if (SubLen > SIZE)
517 |                   SubLen = SIZE;
518 |             iter *= 2;
519 |       }
520 | }
521 |
522 | /*
523 | ** d = a/b by computing the reciprocal of b and then multiplying
524 | ** that by a.  It's faster this way because the normal digit by
525 | ** digit method has N^2 growth, and this method will have the same
526 | ** growth as our multiplication method.
527 | */
528 |
529 | void Divide(Short *d, Short *a, Short *b, Indexer Len)
530 | {
531 |       Reciprocal(Work2, b, Len);
532 |       FastMul(MulWork, a, Work2, Len);
533 |       Round2x(d, MulWork, Len);
534 | }
535 |
536 | /*
537 | ** r = sqrt(n) by computing the reciprocal of the square root of
538 | ** n and then multiplying that by n
539 | */
540 |
541 | void Sqrt(Short *r, Short *n, Indexer Len, Indexer SubLen)
542 | {
543 |       RSqrt(Work2, n, Len, SubLen);
544 |       FastMul(MulWork, n, Work2, Len);
545 |       Round2x(r, MulWork, Len);
546 | }
547 |
548 | void usage(void)
549 | {
550 |       puts("This program requires the number of digits/4 to calculate");
551 |       puts("This number must be a power of two.");
552 |       exit(-1);
553 | }
554 |
555 | int main(int argc,char *argv[])
556 | {
557 |       Indexer x;
558 |       Indexer SubLen;
559 |       double Pow2, tempd, carryd;
560 |       int AGMIter;
561 |       int NeededIter;
562 |       clock_t T0, T1, T2, T3;
563 |
564 |       if (argc < 2)
565 |             usage();
566 |
567 |       SIZE = atoi(argv[1]);
568 |       if (!ispow2(SIZE))
569 |             usage();
570 |
571 |       a = (Short *)malloc(sizeof(Short)*SIZE);
572 |       b = (Short *)malloc(sizeof(Short)*SIZE);
573 |       c = (Short *)malloc(sizeof(Short)*SIZE);
574 |       Work1 = (Short *)malloc(sizeof(Short)*SIZE);
575 |       Work2 = (Short *)malloc(sizeof(Short)*SIZE);
576 |       Work3 = (Short *)malloc(sizeof(Short)*SIZE);
577 |       MulWork = (Short *)malloc(sizeof(Short)*SIZE*4);
578 |
579 |       if ((a ==NULL) || (b == NULL) || (c == NULL) || (Work1 == NULL) ||
580 |           (Work2 == NULL) || (MulWork == NULL))
581 |       {
582 |             printf("Unable to allocate memory\n");exit(1);
583 |       }
584 |
585 |       NeededIter = log(SIZE)/log(2.0) + 1;
586 |       Pow2 = 4.0;
587 |       AGMIter = NeededIter + 1;
588 |       SubLen = 1;
589 |
590 |       T0 = clock();
591 |
592 |       for (x = 0; x < SIZE; x++)
593 |             a[x] = b[x] = c[x] = Work3[x] = 0;
594 |       a[0] = 1;
595 |       Work3[0] = 2;
596 |       RSqrt(b, Work3, SIZE, 1);
597 |       c[0] = 1;
598 |
599 |       T1 = clock();
600 |
```

```
601 |       /*
602 |       printf("AGMIter %d\n", AGMIter);
603 |       PrintShort2("a AGM: ", a, SIZE);
604 |       PrintShort2("b AGM: ", b, SIZE);
605 |       PrintShort2("C sum: ", c, SIZE);
606 |       */
607 |
608 |       while (AGMIter--)
609 |       {
610 |             Sub(Work1, a, b, SIZE);                /* w = (a-b)/2      */
611 |             DivBy2(Work1, SIZE);
612 |             FastMul(MulWork, Work1, Work1, SIZE);  /* m = w*w          */
613 |             Round2x(Work1, MulWork, SIZE);
614 |
615 |             carryd = 0.0;                          /* m = m* w^(J+1)   */
616 |             for (x = SIZE - 1; x >= 0; x--)
617 |             {
618 |                   tempd = Pow2*Work1[x] + carryd;
619 |                   carryd = floor(tempd / 10000.0);
620 |                   Work1[x] = tempd - carryd*10000.0;
621 |             }
622 |             Pow2 *= 2.0;
623 |             Sub(c, c, Work1, SIZE);                /* c = c - m        */
624 |
625 |             /* Save some time on last iter */
626 |
627 |             if (AGMIter != 0)
628 |                   FastMul(MulWork, a, b, SIZE);    /* m = a*b          */
629 |             Add(a, a, b, SIZE);                    /* a = (a+b)/2      */
630 |             DivBy2(a, SIZE);
631 |             if (AGMIter != 0)
632 |             {
633 |                   Round2x(Work3, MulWork, SIZE);
634 |                   Sqrt(b, Work3, SIZE, SubLen); /* b = sqrt(a*b) */
635 |             }
636 |             /*
637 |             printf("AGMIter %d\n", AGMIter);
638 |             PrintShort2("a AGM: ", a, SIZE);
639 |             PrintShort2("b AGM: ", b, SIZE);
640 |             PrintShort2("C sum: ", c, SIZE);
641 |             */
642 |             SubLen *= 2;if (SubLen > SIZE) SubLen = SIZE;
643 |       }
644 |
645 |       T2 = clock();
646 |
647 |       FastMul(MulWork, a, a, SIZE);
648 |       Round2x(a, MulWork, SIZE);
649 |       carryd = 0.0;
650 |       for (x = SIZE - 1; x >= 0; x--)
651 |       {
652 |             tempd = 4.0*a[x] + carryd;
653 |             carryd = floor(tempd / 10000.0);
654 |             a[x] = tempd - carryd*10000.0;
655 |       }
656 |
657 |       Divide(b, a, c, SIZE);
658 |       T3 = clock();
659 |
660 |       PrintShort("\nPI = ", b, SIZE);
661 |
662 |       printf("\nTotal Execution time: %12.4lf\n",
663 |          (double)(T3 - T0) / CLOCKS_PER_SEC);
664 |       printf("Setup time          : %12.4lf\n",
665 |          (double)(T1 - T0) / CLOCKS_PER_SEC);
666 |       printf("AGM Calculation time: %12.4lf\n",
667 |          (double)(T2 - T1) / CLOCKS_PER_SEC);
668 |       printf("Post calc time      : %12.4lf\n",
669 |          (double)(T3 - T2) / CLOCKS_PER_SEC);
670 |
671 |       return 0;
672 | }
```

TEXT STATISTICS

18021 characters
672 lines

LEXICAL STATISTICS

27 comments [std-C]
0 comments [C++]
8 preprocessor instructions
0 constants [character]
25 constants [string]
175 constants [numeric]

SYMBOL TABLE

```
AGMIter         560     587     608     627     631
Add             212     391     396     397     401     403     629
CLOCKS_PER_SEC          663     665     667     669
Copy             77
DivBy2          104     506     611     630
Divide          529     657
DoBorrows       136     392
DoCarries       118     391     396     401     403
FastMul         352     386     394     399     438     446     494     496     504     532     544
        612     628     647
HLen            354     370     373+    377     379+    383+    386+    389     394     395     399+
        400
Indexer          57      59      77      88      90     104     106     153     155     164     166
        212     215     229     232     249     251     269     272     352     354     411     413
        462+    464     529     541+    557     558
Len              77      79+     88      94      96     104     110     153     159     164     173
        212     218     229     235     249     254     269     280     282     286     288     352
        364     366     370     371     387     391     392     396     397     401     402     403
        411     419     442     462     472     529     531     532     533     541     543     544
        545
Long             63     107     112     120     138     214     220+    231     237+    251     271
        290+
Mul             269     366
MulWork          65     438     439     441     443     444     446     447     494     495     496
        497     499     501     502     504     505     532     533     544     545     577     580
        612     613     628     633     647     648
NULL            579+    580+
NeededIter              561     585     587
Negate          249     377     383
NextProd        357     371     386     391     392     394     396     397     399     401     403
Pow2            559     586     618     622
PrintShort              164     660
PrintShort2             153
RSqrt           462     543     596
Reciprocal              411     531
Round2x          88     439     447     495     497     505     533     545     613     633     648
SIZE             66     431     436     437     489     490     492     516     517     567     568
        571     572     573     574     575     576     577     585     592     596     610     611
        612     613     616     623     628     629     630     633     634     642+    647     648
        650     657     660
Short            62      65      77+     88+     91     104     113     118+    136+    153     164
        212+    221     229+    244     249     263     269+    352+    356     357     411+    462+
        529+    541+    571+    572+    573+    574+    575+    576+    577+
SignA           355     373     374     376     390
SignB           355     379     380     382     390
Sqrt            541     634
Sub             229     373     379     392     444     502     610     623
SubLen          413     432     435     436     437     438     439     444     446     447     462
        470     489     490     491     494     495     496     497     500     502     504     505
        506     515     516     517     541     543     558     588     634     642+
T0              562     590     663     665
T1              562     599     665     667
T2              562     645     667     669
T3              562     658     663     669
Work1            65     439     444+    446     495     496     497     502+    504     574     579
        610     611     612+    613     618     620     623
Work2            65     531     532     543     544     575     580
Work3            65     576     593     595     596     633     634
__TURBOC__               56
a        65      77      79      88      99     212     220     229     237     269     274     290
        352     366     373+    394     399     529     532     571     579     593     594     610
        628     629+    630     647+    648     652     654     657
argc            555     564
argv            555     567
atoi            567
b        65      77      79      88      94      98     212     220     229     237     269     274
        284     290     352     366     379+    394     399     529     531     572     579     593
        596     610     628     629     634     657     660
borrow          136     139     141     142     145     231     234     237     238     241     246
        251     253     256     257     260
c        65     573     579     593     597     623+    657
carry            91      93      95      98      99     100     118     122     124     125     128
        214     217     220     222     224
carryd          559     615     618     619     620     649     652     653     654
clock           590     599     645     658
clock_t         562
cur             118     122     124     131     132     136     139     141     148     149
d       231     237     239     242     244     251     256     258     261     263     529     533
dif             229     244
exit            277     552     582
floor           426     428     479     481     619     653
h        50      51      52      53
ia              272     288+    290
ib              272     282+    284     286     290
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  PLAYDEMO.C - demo of background music functions
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include "uclock.h"
 11 | #include "sound.h"
 12 |
 13 | main()
 14 | {
 15 |       int i;
 16 |
 17 |       if (!playb_open(1024))
 18 |       {
 19 |             puts("\aError opening play buffer");
 20 |             return -1;
 21 |       }
 22 |       playb_note(C3, 8);
 23 |       playb_note(REST, 2);
 24 |       playb_note(C3, 8);
 25 |       playb_note(REST, 2);
 26 |       playb_note(E3, 8);
 27 |       playb_note(REST, 2);
 28 |       playb_note(E3, 8);
 29 |       playb_note(REST, 2);
 30 |       playb_note(G3, 8);
 31 |       playb_note(REST, 2);
 32 |       playb_note(G3, 8);
 33 |       playb_note(REST, 2);
 34 |       playb_note(E3, 18);
 35 |       playb_note(REST, 2);
 36 |       for (i = 0; i < 500; ++i)
 37 |             printf("i = %d\n", i);
 38 |       playb_close();
 39 |       return 0;
 40 | }
```

TEXT STATISTICS

```
   845 characters
    40 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     4 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    C3              22      24
    E3              26      28      34
    G3              30      32
    REST            23      25      27      29      31      33      35
    h        9
    i       15      36+     37
    main            13
    playb_close             38
    playb_note              22      23      24      25      26      27      28      29      30      31
            32      33      34      35
    playb_open              17
    printf          37
    puts            19
    stdio            9
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PLAYLIB.C
**
**  Public domain for TC only by Lynn R. Lively
**  Modified by Bob Stout
*/

#include <dos.h>
#include <stdlib.h>
#include "pchwio.h"
#include "sound.h"

#ifndef __ZTC__
 static void (INTERRUPT FAR *n_oldtmint) (void);
#endif

#define TIMER_TICK_INTR 0x1c

static NOTE *     n_buff;
static unsigned   n_buff_sz;
static NOTE *     n_head;
static NOTE *     n_tail;
static unsigned   play_duration;
static unsigned   play_freq;

/*
** Add note to note buff. Return = 1 (note added), 0 (Out of note buff)
*/

int playb_note (unsigned freq, unsigned duration)
{
      if (++n_tail == (n_buff + n_buff_sz))
            n_tail = n_buff;

      if (n_tail == n_head)
      {
            --n_tail;
            return (0);
      }

      n_tail->freq     = freq;
      n_tail->duration = duration;

      return (1);
}

/*
** ISR for background music.
*/

#ifndef __ZTC__
 static void INTERRUPT FAR play_intr (void)
#else
 static int play_intr (struct INT_DATA *idp)
#endif
{
      int_off ();

#ifndef __ZTC__
      (*n_oldtmint) ();                   /* Call Old timer interrupt.  */
#else
      int_prev(idp);
#endif

      if (play_duration == 0)
      {
            soundoff ();

            if (++n_head == (n_buff + n_buff_sz))
                  n_head = n_buff;

            if (n_head == n_tail)
            {
                  --n_head;
                  int_on ();
                  return;
            }

            play_duration = n_head->duration;
            if (0 != (play_freq = n_head->freq))
                  soundon();
            dosound (play_freq);
      }
      else  --play_duration;

      int_on ();

#ifdef __ZTC__
      return 1;                           /* Don't chain                */
#endif
}

/*
** Call this function to init background music. buff_sz is number of
** notes in the note buffer. Returns pointer to buff or NULL if
** out of heap space.
*/
```

```
101 | NOTE * playb_open (unsigned buff_sz)
102 | {
103 |       n_buff =
104 |       n_head =
105 |       n_tail =  (NOTE *) calloc (buff_sz, sizeof (NOTE));
106 |
107 |       if (n_buff != (NOTE *) NULL)
108 |       {
109 |             n_buff_sz     = buff_sz;
110 |
111 |             play_duration =
112 |             play_freq     = 0;
113 |
114 | #ifdef __ZTC__
115 |             int_intercept(TIMER_TICK_INTR, play_intr, 256);
116 | #else
117 |             n_oldtmint    = getvect (TIMER_TICK_INTR);
118 |             setvect (TIMER_TICK_INTR, play_intr);
119 | #endif
120 |       }
121 |       return (n_buff);
122 | }
123 |
124 | /*
125 | ** Return things to normal and free allocated space.
126 | */
127 |
128 | void playb_close (void)
129 | {
130 |       soundoff ();
131 | #ifndef __ZTC__
132 |       setvect (TIMER_TICK_INTR, n_oldtmint);
133 | #else
134 |       int_restore(TIMER_TICK_INTR);
135 | #endif
136 |       free (n_buff);
137 | }
```

TEXT STATISTICS

```
  2829 characters
   137 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    21 preprocessor instructions
     0 constants [character]
     2 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    FAR             16      54
    INTERRUPT       16      54
    INT_DATA        56
    NOTE            21      23      24     101     105+    107
    NULL           107
    TIMER_TICK_INTR         19     115     117     118     132     134
    __ZTC__         15      53      61      90     114     131
    buff_sz        101     105     109
    calloc         105
    dos             10
    dosound         84
    duration        32      44+     81
    free           136
    freq            32      43+     82
    getvect        117
    h       10      11
    idp             56      64
    int_intercept          115
    int_off         59
    int_on          77      88
    int_prev        64
    int_restore            134
    n_buff          21      34      35      71      72     103     107     121     136
    n_buff_sz       22      34      71     109
    n_head          23      37      71      72      74      76      81      82     104
    n_oldtmint              16      62     117     132
    n_tail          24      34      35      37      39      43      44      74     105
    play_duration           25      67      81      86     111
    play_freq       26      82      84     112
    play_intr       54      56     115     118
    playb_close            128
    playb_note              32
    playb_open             101
    setvect        118     132
    soundoff        69     130
    soundon         83
    stdlib          11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  POSIXDIR.C - POSIX-style directory processing
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <string.h>
#include <stdlib.h>
#include <dos.h>
#include <io.h>
#include <errno.h>
#include "sniptype.h"
#include "dirent.h"
#include "filnames.h"         /* For dos2unix() */

#define _NDIRS 20

int DFerr;
DIR _DIRS[_NDIRS];            /* Initialize DIR array to zeros    */

/*****************************************************************/
/*                                                               */
/*  opendir()                                                    */
/*                                                               */
/*  Function: Open a directory for reading.                      */
/*                                                               */
/*  Parameters: 1 - Directory name. May include path spec.       */
/*                                                               */
/*  Returns: Pointer to a DIR typedef'ed structure, similar      */
/*           to fopen() returning a FILE pointer.                */
/*                                                               */
/*           NULL if error, DFerr set as follows:                */
/*           Success_ - No error                                 */
/*           ENOENT   - Could not locate directory or contents   */
/*           ENOTDIR  - Not a directory                          */
/*           ENOMEM   - Too many directories already open        */
/*                                                               */
/*  Side effects: The dd_size element of the DIR structure       */
/*                will contain a number representing the total   */
/*                number of entries in the directory. The        */
/*                dd_loc element will be set to zero since       */
/*                no elements have, as yet, been read.           */
/*                                                               */
/*****************************************************************/

DIR *opendir(char *fname)
{
      int i;
      unsigned n = 0;
      char *p;
      DOSFileData dstruct;

      for (i = 0; i < _NDIRS; ++i)
      {
            if (!_DIRS[i].dd_fd)
                  break;
      }
      if (_NDIRS <= i)
      {
            DFerr = ENOMEM;
            return NULL;
      }

      dos2unix(fname);
      if (':' == fname[1] && 1 < strlen(fname))
            p = &fname[2];
      else  p = fname;
      while ('/' == LAST_CHAR(p) && 1 < strlen(p))
            LAST_CHAR(p) = '\0';

      if (strcmp(p, "/") && strlen(p))
      {
            if (Success_ != (FIND_FIRST(fname, _A_ANY, &_DIRS[i].dd_buf)))
            {
                  DFerr = ENOENT;
                  return NULL;
            }
            if (!(_A_SUBDIR & _DIRS[i].dd_buf.attrib))
            {
                  DFerr = ENOTDIR;
                  return NULL;
            }
      }
      strcpy(_DIRS[i].dd_dirname, fname);
      if (!strlen(p))
            strcat(_DIRS[i].dd_dirname, ".");
      if ('/' != LAST_CHAR(_DIRS[i].dd_dirname))
            strcat(_DIRS[i].dd_dirname, "/");
      strcat(strupr(_DIRS[i].dd_dirname), "*.*");
      if (Success_ != FIND_FIRST(_DIRS[i].dd_dirname,
            _A_ANY, &_DIRS[i].dd_buf))
      {
            DFerr = ENOENT;
```

```
101 |             return NULL;
102 |       }
103 |       memcpy(&dstruct, &_DIRS[i].dd_buf, sizeof(DOSFileData));
104 |       do
105 |       {
106 |             ++n;
107 |       } while (Success_ == FIND_NEXT(&_DIRS[i].dd_buf));
108 |       memcpy(&_DIRS[i].dd_buf, &dstruct, sizeof(DOSFileData));
109 |       _DIRS[i].dd_size = n;
110 |       _DIRS[i].dd_loc  = 0;
111 |       _DIRS[i].dd_fd   = i + 1;
112 |       DFerr = Success_;
113 |       return &_DIRS[i];
114 | }
115 |
116 | /*********************************************************************/
117 | /*                                                                   */
118 | /*  closedir()                                                       */
119 | /*                                                                   */
120 | /*  Function: Close a previously opened directory.                   */
121 | /*                                                                   */
122 | /*  Parameters: 1 - DIR pointer of directory to close.               */
123 | /*                                                                   */
124 | /*  Returns: Success_ or Error_.                                     */
125 | /*                                                                   */
126 | /*********************************************************************/
127 |
128 | int closedir(DIR *dirp)
129 | {
130 |       if (0 == dirp->dd_fd || _NDIRS < dirp->dd_fd)
131 |       {
132 |             DFerr = EBADF;
133 |             return Error_;
134 |       }
135 |       FIND_END(dirp->dd_buf);
136 |       memset(dirp, 0, sizeof(DIR));
137 |       return Success_;
138 | }
139 |
140 | /*********************************************************************/
141 | /*                                                                   */
142 | /*  rewinddir()                                                      */
143 | /*                                                                   */
144 | /*  Function: Reset an open DIR to its first entry.                  */
145 | /*                                                                   */
146 | /*  Parameters: 1 - DIR pointer of directory to rewind.              */
147 | /*                                                                   */
148 | /*  Returns: Success_ or Error_.                                     */
149 | /*                                                                   */
150 | /*********************************************************************/
151 |
152 | int rewinddir(DIR *dirp)
153 | {
154 |       if (0 == dirp->dd_fd || _NDIRS < dirp->dd_fd)
155 |       {
156 |             DFerr = EBADF;
157 |             return Error_;
158 |       }
159 |       FIND_FIRST(dirp->dd_dirname, _A_ANY, &(dirp->dd_buf));
160 |       dirp->dd_loc = 0;
161 |       return Success_;
162 | }
163 |
164 | /*********************************************************************/
165 | /*                                                                   */
166 | /*  seekdir()                                                        */
167 | /*                                                                   */
168 | /*  Function: Point to a selected entry in a DIR.                    */
169 | /*                                                                   */
170 | /*  Parameters: 1 - DIR pointer of directory to rewind.              */
171 | /*              2 - Offset of entry to seek                          */
172 | /*              3 - Origin of offset                                 */
173 | /*                                                                   */
174 | /*  Returns: A DOSFileData pointer, same as returned by DOS find     */
175 | /*           first/next calls.                                       */
176 | /*                                                                   */
177 | /*           NULL if error, DFerr set as follows:                    */
178 | /*           Success_ - No error                                     */
179 | /*           EBADF    - Bad file (DIR) pointer                       */
180 | /*           EACCES   - Illegal origin specification                 */
181 | /*           EOF      - Attempt to seek past end of directory        */
182 | /*                                                                   */
183 | /*********************************************************************/
184 |
185 | DOSFileData *seekdir(DIR *dirp, int offset, int origin)
186 | {
187 |       int i, loc;
188 |
189 |       if (0 == dirp->dd_fd || _NDIRS < dirp->dd_fd)
190 |       {
191 |             DFerr = EBADF;
192 |             return NULL;
193 |       }
194 |       switch (origin)
195 |       {
196 |       case SEEK_SET:
197 |             loc = offset + 1;
198 |             break;
199 |       case SEEK_CUR:
200 |             loc = dirp->dd_loc + offset;
```

```
201 |             break;
202 |       case SEEK_END:
203 |             loc = dirp->dd_size + offset;
204 |             break;
205 |       default:
206 |             DFerr = EACCES;
207 |             return NULL;
208 |       }
209 |
210 |       if (loc > (int)dirp->dd_size || 0 >= loc)
211 |       {
212 |             DFerr = EOF;
213 |             return NULL;
214 |       }
215 |
216 |       rewinddir(dirp);
217 |       for (i = 0; i < loc; ++i)
218 |             readdir(dirp);
219 |
220 |       DFerr = Success_;
221 |       return (&(dirp->dd_buf));
222 | }
223 |
224 | /*****************************************************************/
225 | /*                                                               */
226 | /*  readdir()                                                    */
227 | /*                                                               */
228 | /*  Function: Reads entries from an open directory.              */
229 | /*                                                               */
230 | /*  Parameters: 1 - DIR pointer of directory to read.            */
231 | /*                                                               */
232 | /*  Returns: A DOSFileData pointer, same as returned by DOS find  */
233 | /*           first/next calls.                                   */
234 | /*                                                               */
235 | /*           NULL if error, DFerr set as follows:                */
236 | /*           Success_ - No error                                 */
237 | /*           EBADF    - Bad file (DIR) pointer                   */
238 | /*           EOF      - Attempt to read past end of directory    */
239 | /*                                                               */
240 | /*  Side effects: The dd_loc element of the DIR structure        */
241 | /*                will contain a number representing which       */
242 | /*                element of the directory was returned. It may  */
243 | /*                range from 1 to dd_size.                       */
244 | /*                                                               */
245 | /*****************************************************************/
246 |
247 | DOSFileData *readdir(DIR *dirp)
248 | {
249 |       if (0 == dirp->dd_fd || _NDIRS < dirp->dd_fd)
250 |       {
251 |             DFerr = EBADF;
252 |             return NULL;
253 |       }
254 |       if (0 == dirp->dd_loc || Success_ == FIND_NEXT(&(dirp->dd_buf)))
255 |       {
256 |             dirp->dd_loc += 1;
257 |             DFerr = Success_;
258 |             return (&(dirp->dd_buf));
259 |       }
260 |       else
261 |       {
262 |             DFerr = EOF;
263 |             return NULL;
264 |       }
265 | }
```

TEXT STATISTICS

```
 10407 characters
   265 lines
```

LEXICAL STATISTICS

```
    92 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     4 constants [character]
     7 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    DFerr            25      67      82      87     100     112     132     156     191     206     212
           220      251     257     262
    DIR              26      53     128     136     152     185     247
    DOSFileData              58     103     108     185     247
    EACCES          206
    EBADF           132     156     191     251
    ENOENT           82     100
    ENOMEM           67
    ENOTDIR          87
    EOF             212     262
    Error_          133     157
    FIND_END        135
    FIND_FIRST               80      97     159
    FIND_NEXT       107     254
    LAST_CHAR        75      76      94
    NULL             68      83      88     101     192     207     213     252     263
    SEEK_CUR        199
    SEEK_END        202
    SEEK_SET        196
    Success_         80      97     107     112     137     161     220     254     257
    _A_ANY           80      98     159
    _A_SUBDIR        85
    _DIRS            26      62      80      85      91      93      94      95      96      97      98
           103      107     108     109     110     111     113
    _NDIRS           23      26      60      65     130     154     189     249
    attrib           85
    closedir        128
    dd_buf           80      85      98     103     107     108     135     159     221     254     258
    dd_dirname               91      93      94      95      96      97     159
    dd_fd            62     111     130+    154+    189+    249+
    dd_loc          110     160     200     254     256
    dd_size         109     203     210
    dirp            128     130+    135     136     152     154+    159+    160     185     189+    200
           203      210     216     218     221     247     249+    254+    256     258
    dos              16
    dos2unix         71
    dstruct          58     103     108
    errno            18
    fname            53      71      72+     73      74      80      91
    h       14       15      16      17      18
    i       55      60+     62      65      80      85      91      93      94      95      96      97
           98      103     107     108     109     110     111+    113     187     217+
    io               17
    loc             187     197     200     203     210+    217
    memcpy          103     108
    memset          136
    n       56      106     109
    offset          185     197     200     203
    opendir          53
    origin          185     194
    p       57       73      74      75+     76      78+     92
    readdir         218     247
    rewinddir       152     216
    seekdir         185
    stdlib           15
    strcat           93      95      96
    strcmp           78
    strcpy           91
    string           14
    strlen           72      75      78      92
    strupr           96
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  POSIX_LS.C - Directory lister using POSIX style processing
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  This subset version is functionally identical to the
**  version originally published by the author in Tech Specialist
**  magazine and is hereby donated to the public domain.
*/

#include <stdio.h>
#include "dirent.h"
#include "sniptype.h"

void dumpdir(char *dirname, char *mask)
{
      DIR *dirp;
      DOSFileData *dstruct;

      dirp = opendir(dirname);
      if (!dirp)
      {
            printf("Opening %s returned NULL\n\n", dirname);
            return;
      }
      printf("Dir %s has %d entries\n", dirname, dirp->dd_size);
      do
      {
            if (NULL != (dstruct = readdir(dirp)))
            {
                  if (Success_ == dirmask(dstruct, mask, NULL, _A_ANY, 0))
                        printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
            }
            else  puts("EOF\n");
      } while (dstruct);
      printf("seekdir( -1) returned %p\n", seekdir(dirp, -1, SEEK_SET));
      printf("seekdir(999) returned %p\n", seekdir(dirp, 999, SEEK_SET));
      printf("seekdir(0, SEEK_SET) returned %p\n", dstruct = seekdir(dirp,
            0, SEEK_SET));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(1, SEEK_SET) returned %p\n", dstruct = seekdir(dirp,
            1, SEEK_SET));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(4, SEEK_SET) returned %p\n", dstruct = seekdir(dirp,
            4, SEEK_SET));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(4, SEEK_CUR) returned %p\n", dstruct = seekdir(dirp,
            4, SEEK_CUR));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(-1, SEEK_CUR) returned %p\n", dstruct = seekdir(dirp,
            -1, SEEK_CUR));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(1, SEEK_CUR) returned %p\n", dstruct = seekdir(dirp,
            1, SEEK_CUR));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(0, SEEK_END) returned %p\n", dstruct = seekdir(dirp,
            0, SEEK_END));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(-1, SEEK_END) returned %p\n", dstruct = seekdir(dirp,
            -1, SEEK_END));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      printf("seekdir(-4, SEEK_END) returned %p\n", dstruct = seekdir(dirp,
            -4, SEEK_END));
      printf("%3d - %s\n", dirp->dd_loc, ff_name(dstruct));
      closedir(dirp);
}

main(int argc, char *argv[])
{
      char *mask = NULL, *dirname;

      if (1 < argc)
            dirname = argv[1];
      else  dirname = ".";

      if (2 < argc)
            mask = argv[2];
      printf("Calling dumpdir(%s, %s)\n\n", dirname, mask);
      dumpdir(dirname, mask);
      return 0;
}
```

TEXT STATISTICS

```
  3085 characters
    84 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    28 constants [string]
    17 constants [numeric]
```

SYMBOL TABLE

```
    DIR             20
    DOSFileData             21
    NULL            32      34      73
    SEEK_CUR        51      54      57
    SEEK_END        60      63      66
    SEEK_SET        39      40      42      45      48
    Success_        34
    _A_ANY          34
    argc            71      75      79
    argv            71      76      80
    closedir        68
    dd_loc          35      43      46      49      52      55      58      61      64      67
    dd_size         29
    dirmask         34
    dirname         18      23      26      29      73      76      77      81      82
    dirp            20      23      24      29      32      35      39      40      41      43      44
            46      47      49      50      52      53      55      56      58      59      61      62
            64      65      67      68
    dstruct         21      32      34      35      38      41      43      44      46      47      49
            50      52      53      55      56      58      59      61      62      64      65      67
    dumpdir         18      82
    ff_name         35      43      46      49      52      55      58      61      64      67
    h       14
    main            71
    mask            18      34      73      80      81      82
    opendir         23
    printf          26      29      35      39      40      41      43      44      46      47      49
            50      52      53      55      56      58      59      61      62      64      65      67
            81
    puts            37
    readdir         32
    seekdir         39      40      41      44      47      50      53      56      59      62      65
    stdio           14
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
      This program is similar to a program of the same name found on UNIX.
      It prints the files named in the command tail with headings
      except as modified below.

      usage: pr [-i -ln -on -pname -tn -wn] file1[ file2 ... filen]
      where:      -i     = accept files from stdin
                  -ln    = set lines per page to n
                  -on    = set page offset to n
                  -pname = output to file <name>
                  -tn    = set tabs to n cols
                  -wn    = set page width to n

      note: the expr 'PAGE(mesg)' found in col 1 will cause a formfeed
                  and the 'mesg' to be included in the title line on this and
                  each subsequent page until EOF or another PAGE.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "getopts.h"                          /* Also in SNIPPETS     */

#define TAB_DEFAULT 4
#define PAGE_LENGTH 60
#define PAGE_OFFSET 0
#define PAGE_WIDTH      80
#define MAX_ARGS    70
#define MAX_FILES 64
#define PATH_LENGTH 63
#define PAGE(head)


int       page_length = PAGE_LENGTH;
int       page_offset = PAGE_OFFSET;
int       page_width  = PAGE_WIDTH;
int       tab_width   = TAB_DEFAULT;
Boolean_T read_stdin  = False_;

char      print_name[FILENAME_MAX] = "PRN";
char      filenames[MAX_FILES] [FILENAME_MAX];

struct Option_Tag options[] = {
      {'I',  False_, Boolean_Tag, &read_stdin  , NULL, NULL, NULL},
      {'L',  False_, Word_Tag,    &page_length , NULL, NULL, NULL},
      {'O',  False_, Word_Tag,    &page_offset , NULL, NULL, NULL},
      {'T',  False_, Word_Tag,    &tab_width   , NULL, NULL, NULL},
      {'W',  False_, Word_Tag,    &page_width  , NULL, NULL, NULL},
      {'P',  False_, String_Tag,   print_name  , NULL, NULL, NULL},
      {'\0', False_, Error_Tag,    NULL        , NULL, NULL, NULL}
};

char title[80];
char Date[20];
char Time[20];
int  ln, pn;

void prt (char fnp[], FILE *lp);
void new_page (char *fnp, FILE *lp);
void indent(FILE *lp);

PAGE (MAIN)
main(int argc, char *argv[])  /* copy file to printer */
{
      FILE *lp;
      int fi;
      int pn;
      char *cp;

      if (argc < 2) /* No args so tell 'em how it works */
      {
            fprintf(stderr,
            "usage:\n\npr %s %s\n\n",
            "[-i] [-lnn] [-onn] [-p<name>] [-tn] [-wnn]",
            "[file1[ file2 ... filen]]");
            fprintf(stderr,
            "where: i = read 'stdin' for filenames to print\n");
            fprintf(stderr,
            "       l = lines-per-page and nn <= 120\n");
            fprintf(stderr,
            "       o = page offset    and nn <= 120\n");
            fprintf(stderr,
            "       p = print redirection and\n");
            fprintf(stderr,
            "           <name> = pathname or devicename\n");
            fprintf(stderr,
            "       t = spaces-per-tab and n  <= 8\n");
            fprintf(stderr,
            "       w = page width     and nn <= 160\n\n");
            fprintf(stderr,
            "Notes: PAGE(<title text of any length>) in col 1 of text file\n");
            fprintf(stderr,
            "       and <title text...> the title you want.\n\n");
            fprintf(stderr,
            "       C pgms should include the following macro:\n\n");
            fprintf(stderr,
            "           #define PAGE(title)\n\n");
```

```
101 |             fprintf(stderr,
102 |             "       < and > not required and should not be used\n\n");
103 |             exit(0);
104 |       }
105 |
106 |       if (Error_ == getopts(argc, argv))
107 |             return 1;
108 |
109 |       if ((page_length <= 0) || (page_length > 120))
110 |             page_length = PAGE_LENGTH;
111 |
112 |       if ((tab_width <= 0) || (tab_width > 8))
113 |             tab_width = TAB_DEFAULT;
114 |
115 |       if ((page_offset < 0) || (page_offset > 120))
116 |             page_offset = PAGE_OFFSET;
117 |
118 |       if ((page_width <= 0) || (page_width > 160))
119 |             page_width = PAGE_WIDTH;
120 |
121 |       for (fi = 0, pn = 1; pn < xargc; ++fi, ++pn)
122 |       {
123 |             if (fi < MAX_FILES)
124 |                   strcpy(filenames[fi], xargv[pn]);
125 |             else
126 |             {
127 |                   fprintf(stderr, "pr: "
128 |                         "Exceeded maximum file capacity\n");
129 |                   break;
130 |             }
131 |       }
132 |
133 |       if ((lp = fopen(print_name, "w")) == 0)
134 |       {
135 |             fprintf(stderr, "pr: Unable to open %s as output\n", print_name);
136 |             exit(1);
137 |       }
138 |
139 |       if (read_stdin)
140 |       {
141 |             for(;;)
142 |             {
143 |                   if (fi == MAX_FILES)
144 |                   {
145 |                         fputs("pr: Exceeded maximum file capacity\n",
146 |                               stderr);
147 |                         break;
148 |                   }
149 |                   cp = fgets(filenames [fi], FILENAME_MAX, stdin);
150 |                   if (!cp)
151 |                         break;
152 |                   else  fi++;
153 |             }
154 |       }
155 |       /* now print each file */
156 |
157 |       for (pn = 0; pn < fi; pn++)
158 |             prt(filenames [pn], lp);  /* print the file */
159 |       return 0;
160 | }
161 | PAGE (NEW PAGE)
162 |
163 | void new_page (char *fnp, FILE *lp)
164 | {
165 |       if (ln < 3)
166 |             return;
167 |       ++pn;
168 |       if (pn > 1)
169 |             fputc('\f', lp);
170 |       fprintf(lp, "%s    %s %s    PAGE %d:  %s\n\n",
171 |                   fnp, Date, Time, pn, title);
172 |       ln = 2;
173 | }
174 |
175 | PAGE (PRINT FILE)
176 | void prt (char fnp[], FILE *lp)
177 | {
178 |       FILE *inp_file;
179 |       int col;
180 |       char line [256], *st, *et, *sp;
181 |       time_t now;
182 |       struct tm *tnow;
183 |
184 |       inp_file = fopen(fnp, "r");
185 |       if (!inp_file)
186 |       {
187 |             fprintf(stderr, "pr: unable to open %s\n", fnp);
188 |             return;
189 |       }
190 |       else
191 |             fprintf(stderr, "pr: printing %s\n", fnp);
192 |
193 |       pn = 0;
194 |       ln = 999;
195 |       now  = time(&now);
196 |       tnow = localtime(&now);
197 |       strftime(Date, 19, "%a %d %b %Y", tnow);
198 |       strftime(Time, 19, "%I:%M %PM", tnow);
199 |       *title = '\0';
200 |
```

```
201 |       while (fgets(line, 256, inp_file))
202 |       {
203 |             if (strncmp(line, "PAGE", 4) == 0)
204 |             {
205 |                   if (NULL != (st = strchr(line, '(')))
206 |                   {
207 |                         et = strchr(line, ')');
208 |                         strncpy(title, st + 1, (et) ? et - st - 1 : 160);
209 |                   }
210 |                   ln = 999;
211 |             }
212 |
213 |             if (ln > page_length)
214 |                   new_page(fnp, lp);
215 |
216 |             if (page_offset)
217 |                   indent(lp);
218 |
219 |             for (col = (page_offset) ? page_offset : 1, sp = &line[0];
220 |                    *sp; sp++)
221 |             {
222 |                   switch (*sp)
223 |                   {
224 |                   case '\t':  /* tab character */
225 |                         do
226 |                         {
227 |                               fputc(' ', lp);
228 |                               col++;
229 |                               if (col > page_width)
230 |                               {
231 |                                     fputc('\n', lp);
232 |                                     col = (page_offset) ? page_offset : 1;
233 |                                     ln++;
234 |                                     if (ln > page_length)
235 |                                           new_page(fnp, lp);
236 |                                     if (page_offset)
237 |                                           indent(lp);
238 |                                     break;
239 |                               }
240 |                         } while ((col - 1) % tab_width);
241 |                         break;
242 |
243 |                   case '\f':  /* form feed character */
244 |                         new_page(fnp, lp);
245 |                         break;
246 |
247 |                   default:
248 |                         fputc(*sp, lp);
249 |                         ++col;
250 |                         if (col > page_width)
251 |                         {
252 |                               fputc('\n', lp);
253 |                               col = (page_offset) ? page_offset - 1 : 0;
254 |                               ln++;
255 |                               if (ln > page_length)
256 |                                     new_page(fnp, lp);
257 |                               if (page_offset)
258 |                                     indent(lp);
259 |                         }
260 |                   }
261 |             } /* of line print (for) */
262 |             ++ln;
263 |       } /* of while not eof */
264 |       fclose(inp_file);
265 |       fputc(014, lp);
266 | } /* of print */
267 |
268 | void indent(FILE *lp)
269 | {
270 |       int i;
271 |
272 |       for(i = 1; i < page_offset; i++)
273 |             fputc(' ', lp);
274 | }
```

```
TEXT STATISTICS

  8671 characters
   274 lines

LEXICAL STATISTICS

    12 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
    17 constants [character]
    29 constants [string]
    50 constants [numeric]

SYMBOL TABLE

    Boolean_T       41
    Boolean_Tag             47
    Date             57     171     197
    Error_          106
    Error_Tag        53
    FILE             61      62      63      68     163     175     176     178     268
    FILENAME_MAX            43      44     149
    False_           41      47      48      49      50      51      52      53
    MAIN             65
    MAX_ARGS         31
    MAX_FILES        32      44     123     143
    NEW             161
    NULL             47+     48+     49+     50+     51+     52+     53+    205
    Option_Tag              46
    PAGE             34      65     161+    175
    PAGE_LENGTH             28      37     110
    PAGE_OFFSET             29      38     116
    PAGE_WIDTH              30      39     119
    PATH_LENGTH             33
    PRINT           175
    String_Tag              52
    TAB_DEFAULT             27      40     113
    Time             58     171     198
    Word_Tag         48      49      50      51
    argc             66      73     106
    argv             66     106
    col             179     219     228     229     232     240     249     250     253
    cp               71     149     150
    et              180     207     208+
    exit            103     136
    fclose          264
    fgets           149     201
    fi               69     121+    123     124     143     149     152     157
    filenames        44     124     149     158
    fnp              61      62     163     171     176     184     187     191     214     235     244
           256
    fopen           133     184
    fprintf          75      79      81      83      85      87      89      91      93      95      97
           99       101     127     135     170     187     191
    fputc           169     227     231     248     252     265     273
    fputs           145
    getopts         106
    h        21      22      23      24
    head             34
    i       270     272+
    indent           63     217     237     258     268
    inp_file        178     184     185     201     264
    line            180     201     203     205     207     219
    ln               59     165     172     194     210     213     233     234     254     255     262
    localtime       196
    lp               61      62      63      68     133     158     163     169     170     176     214
           217      227     231     235     237     244     248     252     256     258     265     268
           273
    main             66
    new_page         62     163     214     235     244     256
    now             181     195+    196
    options          46
    page_length              37      48     109+    110     213     234     255
    page_offset              38      49     115+    116     216     219+    232+    236     253+    257
           272
    page_width               39      51     118+    119     229     250
    pn               59      70     121+    124     157+    158     167     168     171     193
    print_name               43      52     133     135
    prt              61     158     176
    read_stdin               41      47     139
    sp              180     219     220+    222     248
    st              180     205     208+
    stderr           75      79      81      83      85      87      89      91      93      95      97
           99       101     127     135     146     187     191
    stdin           149
    stdio            21
    stdlib           22
    strchr          205     207
    strcpy          124
    strftime        197     198
    string           23
    strncmp         203
    strncpy         208
    tab_width        40      50     112+    113     240
    time             24     195
    time_t          181
    title            56     171     199     208
    tm              182
    tnow            182     196     197     198
```

```
xargc           121
xargv           124
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* printq.c 12-22-91 Robert Mashlan, Public Domain
  4 |
  5 |    A small program that utilizes the prnspool module,
  6 |    which is an interface to the DOS program PRINT.COM
  7 |
  8 | */
  9 |
 10 | #include "prnspool.h"
 11 | #include <stdio.h>
 12 | #include <string.h>
 13 |
 14 | int main(int argc, char **argv )
 15 | {
 16 |       char far *files;
 17 |       int i;
 18 |       int addfiles = 1;
 19 |
 20 |       if (!printspool_installed())
 21 |       {
 22 |             printf("print.com not installed\n");
 23 |             return 0;
 24 |       }
 25 |       for (i = 1; i < argc; i++)
 26 |       {
 27 |             if (stricmp(argv[i],"/T") == 0)
 28 |                   printspool_cancel();    /* cancel all files in queue */
 29 |             else if (stricmp(argv[i],"/C") == 0)
 30 |                   addfiles = 0;           /* cancel all listed files */
 31 |             else if (stricmp(argv[i],"/P") == 0)
 32 |                   addfiles = 1;           /* add all listed files */
 33 |             else
 34 |             /* here the specified file should really have the full pathname */
 35 |             {
 36 |                   if (addfiles)
 37 |                         printspool_submit(argv[i]);
 38 |                   else  printspool_remove(argv[i]);
 39 |                   if (printspool_errno)
 40 |                         puts(printspool_errlist[printspool_errno]);
 41 |             }
 42 |       }
 43 |       printf("files currently in queue:\n");
 44 |       for (files = printspool_getqueue(); *files; files += 64)
 45 |             printf("\t%Fs\n", files);
 46 |       printspool_endhold();
 47 |       return 0;
 48 | }
```

TEXT STATISTICS

```
  1501 characters
    48 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     7 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    addfiles        18      30      32      36
    argc            14      25
    argv            14      27      29      31      37      38
    far             16
    files           16      44+     45
    h       11      12
    i       17      25+     27      29      31      37      38
    main            14
    printf          22      43      45
    printspool_cancel       28
    printspool_endhold              46
    printspool_errlist              40
    printspool_errno        39      40
    printspool_getqueue             44
    printspool_installed            20
    printspool_remove       38
    printspool_submit       37
    puts            40
    stdio           11
    stricmp         27      29      31
    string          12
```

```
/* +++Date last modified: 05-Jul-1997 */

/* prnspool.c 12-22-91 Robert Mashlan, public domain */
/* DOS print spooler interface functions             */

#include <stdio.h>
#include <dos.h>
#include "prnspool.h"

int printspool_errno = 0;

char *printspool_errlist[] = {
      "No error",
      "Function Invalid",
      "File not found",
      "Path not found",
      "Too many open files",
      "Access denied",
      "",
      "",
      "Queue full",
      "Spooler busy",
      "",
      "",
      "Name too long",
      "",
      "Drive invalid"
      };

/* returns -1 if printspooler installed */
/* 0 otherwise                          */

int printspool_installed(void)
{
      union REGS r;

      r.x.ax = 0x0100;
      int86(0x2f, &r, &r);
      if(r.h.al==0xff)
      {
            printspool_errno=0;
            return -1;
      }
      else
      {
            printspool_errno=1;
            return 0;
      }
}

/* submits a file name to be printed */
/* returns error code                */

int printspool_submit( const char *pathname )
{
      struct PACKET packet;
      union REGS r;
      struct SREGS s;

      packet.level = 0;
      packet.pathname = (char FAR *)pathname;
      s.ds   = FP_SEG(&packet);
      r.x.dx = FP_OFF(&packet);
      r.x.ax = 0x0101;
      int86x(0x2f, &r, &r, &s);
      if(!r.x.cflag)
            return printspool_errno = 0;
      else  return printspool_errno = r.x.ax;
}

/* removes a file from the print queue */

int printspool_remove( const char FAR *fname )
{
      union REGS r;
      struct SREGS s;

      s.ds   = FP_SEG(fname);
      r.x.dx = FP_OFF(fname);
      r.x.ax = 0x0102;
      int86x(0x2f, &r, &r, &s);
      if(!r.x.cflag)
            return printspool_errno = 0;
      else  return printspool_errno=r.x.ax;
}

/* cancels all files in the print queue */

int printspool_cancel(void)
{
      union REGS r;

      r.x.ax = 0x0103;
      int86(0x2f, &r, &r);
      if(!r.x.cflag)
            return printspool_errno = 0;
      else  return printspool_errno = r.x.ax;
}

/* ends hold state after a call to printspool_getqueue */
```

```
101 | /* or printspool_errorcount                              */
102 |
103 | void printspool_endhold(void)
104 | {
105 |       union REGS r;
106 |
107 |       r.x.ax = 0x0105;
108 |       int86(0x2f, &r, &r);
109 | }
110 |
111 | /* returns a far pointer to the printspooler queue, */
112 | /* an array of 64 char asciiz strings                */
113 |
114 | char FAR *printspool_getqueue(void)
115 | {
116 |       char FAR *result;
117 |       union REGS r;
118 |       struct SREGS s;
119 |
120 |       r.x.ax = 0x0104;
121 |       int86x(0x2f, &r, &r, &s);
122 |       result = MK_FP(s.ds,r.x.si);
123 |       if (!r.x.cflag)
124 |       {
125 |             printspool_errno = 0;
126 |             return result;
127 |       }
128 |       else
129 |       {
130 |             printspool_errno = r.x.ax;
131 |             return NULL;
132 |       }
133 | }
134 |
135 | /* returns the error count from the printspooler */
136 |
137 | int printspool_errorcount(void)
138 | {
139 |       union REGS r;
140 |
141 |       r.x.ax = 0x0104;
142 |       int86(0x2f, &r, &r);
143 |       if (!r.x.cflag)
144 |       {
145 |             printspool_errno = 0;
146 |             return r.x.dx;          /* return the number of errors */
147 |       }
148 |       else
149 |       {
150 |             printspool_errno = r.x.ax;
151 |             return r.x.dx;
152 |       }
153 | }
```

```
TEXT STATISTICS

  3317 characters
   153 lines

LEXICAL STATISTICS

    15 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    16 constants [string]
    26 constants [numeric]

SYMBOL TABLE

    FAR             61      73     114     116
    FP_OFF          63      79
    FP_SEG          62      78
    MK_FP          122
    NULL           131
    PACKET          56
    REGS            35      57      75      91     105     117     139
    SREGS           58      76     118
    al              39
    ax              37      64      68      80      84      93      97     107     120     130     141
          150
    cflag           66      82      95     123     143
    dos              7
    ds              62      78     122
    dx              63      79     146     151
    fname           73      78      79
    h        6       7      39
    int86           38      94     108     142
    int86x          65      81     121
    level           60
    packet          56      60      61      62      63
    pathname        54      61+
    printspool_cancel       89
    printspool_endhold             103
    printspool_errlist              12
    printspool_errno        10      41      46      67      68      83      84      96      97     125
          130     145     150
    printspool_errorcount          137
    printspool_getqueue            114
    printspool_installed            33
    printspool_remove       73
    printspool_submit       54
    r       35      37      38+     39      57      63      64      65+     66      68      75      79
            80      81+     82      84      91      93      94+     95      97     105     107     108+
          117     120     121+    122     123     130     139     141     142+    143     146     150
          151
    result         116     122     126
    s       58      62      65      76      78      81     118     121     122
    si             122
    stdio            6
    x       37      63      64      66      68      79      80      82      84      93      95      97
          107     120     122     123     130     141     143     146     150     151
```

```
/* +++Date last modified: 05-Jul-1997 */

/* prnspool.h 12-22-91 Robert Mashlan, public domain */
/* print spooler interface functions header file     */
/* modified by Bob Stout, Nov '93                    */

#ifndef PRNSPOOL__H
#define PRNSPOOL__H

#include "mk_fp.h"      /* Also includes extkword.h   */

#ifndef __TURBOC__
 #if (defined(_MSC_VER) && (_MSC_VER >= 700)) || (defined(__SC__))
  /* Make FP_xxx macros lvalues as in older versions */
  #undef FP_SEG
  #undef FP_OFF
  #define FP_SEG(fp)    ((unsigned)((unsigned long)(fp) >> 16))
  #define FP_OFF(fp)    ((unsigned)(fp && 0xffff))
 #endif
#endif

struct PACKET {
      unsigned char level;
      char far *pathname;
};

extern int   printspool_errno;
extern char *printspool_errlist[];

int  printspool_installed(void);
int  printspool_submit( const char *pathname );
int  printspool_remove( const char far *fname );
int  printspool_cancel(void);
char far *printspool_getqueue(void);
void printspool_endhold(void);
int  printspool_errorcount(void);

#define PSENOERR   0x00
#define PSEINVFNC  0x01
#define PSENOFILE  0x02
#define PSENOPATH  0x03
#define PSEMFILE   0x04
#define PSEACCES   0x05
#define PSEQUEFUL  0x08
#define PSESPLBUSY 0x09
#define PSENME2LNG 0x0c
#define PSEINVDRV  0x0f

#endif /* PRNSPOOL__H */
```

```
TEXT STATISTICS

  1333 characters
    49 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
    22 preprocessor instructions
     0 constants [character]
     1 constants [string]
    13 constants [numeric]

SYMBOL TABLE

    FP_OFF          16      18
    FP_SEG          15      17
    PACKET          22
    PRNSPOOL__H              7       8
    PSEACCES        43
    PSEINVDRV       47
    PSEINVFNC       39
    PSEMFILE        42
    PSENME2LNG              46
    PSENOERR        38
    PSENOFILE       40
    PSENOPATH       41
    PSEQUEFUL       44
    PSESPLBUSY              45
    _MSC_VER        13+
    __SC__          13
    __TURBOC__              12
    defined         13+
    far             24      32      34
    fname           32
    fp              17+     18+
    level           23
    pathname        24      31
    printspool_cancel       33
    printspool_endhold              35
    printspool_errlist              28
    printspool_errno        27
    printspool_errorcount           36
    printspool_getqueue             34
    printspool_installed            30
    printspool_remove       32
    printspool_submit       31
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  PRNTSELF.C - A program which prints its own source
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include <string.h>
 11 |
 12 | #ifdef __WATCOMC__
 13 |  #pragma off (unreferenced);
 14 | #endif
 15 |
 16 | #ifdef __TURBOC__
 17 |  #pragma argsused
 18 | #endif
 19 |
 20 | main(int argc, char *argv[])
 21 | {
 22 |       FILE *in;
 23 |       char fname[13], *ptr;
 24 |       char line[1024];        /* Nice & roomy   */
 25 |
 26 |       /*
 27 |       ** Get the source name by replacing the executable's COM or EXE with C
 28 |       */
 29 |
 30 |       strcpy(fname, argv[0]);
 31 |       ptr = strrchr(fname, '.');
 32 |       strcpy(++ptr, "C");
 33 |
 34 |       /*
 35 |       ** Print its own source
 36 |       */
 37 |
 38 |       if (NULL != (in = fopen(fname, "r")))
 39 |       {
 40 |             while (NULL != fgets(line, 1024, in))
 41 |                   fputs(line, stdprn);
 42 |             return 0;
 43 |       }
 44 |       else  return -1;
 45 | }
```

TEXT STATISTICS

```
   886 characters
    45 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     1 constants [character]
     2 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    FILE            22
    NULL            38      40
    __TURBOC__              16
    __WATCOMC__             12
    argc            20
    argsused        17
    argv            20      30
    fgets           40
    fname           23      30      31      38
    fopen           38
    fputs           41
    h       9       10
    in              22      38      40
    line            24      40      41
    main            20
    off             13
    ptr             23      31      32
    stdio            9
    stdprn          41
    strcpy          30      32
    string          10
    strrchr         31
    unreferenced            13
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  prtoggle()
  5 | **
  6 | **  Tee's all standard output to the printer.
  7 | **
  8 | **  Parameters: None
  9 | **
 10 | **  Returns:  0 if operation was successful.
 11 | **           -1 if stdout or stdin is redirected.
 12 | **
 13 | **  Side effects: Flushes the keyboard buffer
 14 | **
 15 | **  Original Copyright 1988-1991 by Bob Stout as part of
 16 | **  the MicroFirm Function Library (MFL)
 17 | **
 18 | **  The user is granted a free limited license to use this source file
 19 | **  to create royalty-free programs, subject to the terms of the
 20 | **  license restrictions specified in the LICENSE.MFL file.
 21 | */
 22 |
 23 | #include <stdio.h>
 24 | #include <stdlib.h>
 25 | #include <conio.h>
 26 | #include <io.h>
 27 | #include "sniptype.h"
 28 | #include "sniprint.h"
 29 | #include "snipkbio.h"
 30 |
 31 | int prtoggle(void)
 32 | {
 33 |       if (!isatty(fileno(stdin)) || !isatty(fileno(stdout)))
 34 |             return -1;
 35 |       while (kbhit())           /* Flush the keyboard buffer            */
 36 |             getch();
 37 |       ungetkey('P' - 64);       /* Stuff a Ctrl-P into the buffer       */
 38 |       system("");               /* Let COMMAND.COM do the work          */
 39 |       return 0;
 40 | }
```

TEXT STATISTICS

```
  1134 characters
    40 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     1 constants [character]
     4 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    conio           25
    fileno          33+
    getch           36
    h       23      24      25      26
    io              26
    isatty          33+
    kbhit           35
    prtoggle        31
    stdin           33
    stdio           23
    stdlib          24
    stdout          33
    system          38
    ungetkey        37
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PRTSC.C - Access the BIOS print screen function
**
**  public domain demo by Bob Stout
*/

#include <dos.h>
#include "extkword.h"
#include "sniprint.h"

/*
**  Get screen printing status
**
**  0 - Ready
**  1 - Screen printing in process
**  2 - Error occurred last time
*/

int PrtScrnStat(void)
{
      return ((int)*((char FAR *)(0x00500000L)));
}

/*
**  Print the current screen
*/

int PrtScrn(void)
{

      union REGS regs;                    /* Dummy for use by int86()  */

      if (1 == PrtScrnStat())             /* Can we print now?          */
            return -1;                    /* Nope, return with error    */
      int86(5, &regs, &regs);            /* Issue Int 5                */
      return 0;
}

#ifdef TEST

#include <stdio.h>

main()
{
      printf("PrtScrn() returned %d\n", PrtScrn());
      printf("PrtScrnStat() returned %d\n", PrtScrnStat());
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1020 characters
    52 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     4 constants [string]
     6 constants [numeric]

SYMBOL TABLE

    FAR             23
    PrtScrn         30      47
    PrtScrnStat             21      35      48
    REGS            33
    TEST            41
    dos              9
    h        9      43
    int86           37
    main            45
    printf          47      48
    regs            33      37+
    stdio           43
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  PRTSTAT.C - Determine printer status
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #include "sniprint.h"
 10 |
 11 | #include <dos.h>
 12 |
 13 | /*
 14 | **  prtstat() - Call with printer number (0 = LPT1, 1 = LPT2, 2 = LPT3)
 15 | **
 16 | **  Returns status which can be mapped to a PrStatus struct
 17 | */
 18 |
 19 | int prtstat(unsigned int printer_no)
 20 | {
 21 |       union REGS regs;
 22 |
 23 |       regs.h.ah = 2;
 24 |       regs.x.dx = printer_no;
 25 |       int86(0x17, &regs, &regs);
 26 |       return regs.h.ah;
 27 | }
 28 |
 29 | #ifdef TEST
 30 |
 31 | #include <stdio.h>
 32 |
 33 | #define show(x) printf(#x" is %strue (LPT1)\n", mystat.x ? "" : "not ");
 34 |
 35 | main()
 36 | {
 37 |       struct PrStatus mystat;
 38 |
 39 |       *((int *)&mystat) = prtstat(0);
 40 |       show(notbusy);
 41 |       show(selected);
 42 |       show(paperout);
 43 |       return 0;
 44 | }
 45 |
 46 | #endif /* TEST */
```

TEXT STATISTICS

```
   814 characters
    46 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     3 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    LPT1            33
    PrStatus        37
    REGS            21
    TEST            29
    ah              23      26
    dos             11
    dx              24
    h       11      23      26      31
    int86           25
    is              33
    main            35
    mystat          33      37      39
    n"              33
    notbusy         40
    paperout        42
    printer_no              19      24
    printf          33
    prtstat         19      39
    regs            21      23      24      25+     26
    selected        41
    show            33      40      41      42
    stdio           31
    strue           33
    x       24      33+
    x"              33
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  PushDir() and PopDir()
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "dosfiles.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

#define DIR_STACK_SIZE  8
#define MAX_FLEN        67

static int  PushDir_stack_ptr;
static char PushDir_stack[DIR_STACK_SIZE][MAX_FLEN];

/*
**  PushDir()
**
**  Like chdir(), except a drive may be specified and the old directory
**  is saved.
**
**  Arguments: 1 - newdir, the buffer containing the new directory name
**
**  Returns:  -1 - stack overflow
**             0 - error
**             1 - success, still on same drive
**             2 - success, changed drive
**
**  Side effects: Converts name in newdir to upper case and prepends
**                a drive letter.
**
**  CAUTION: Since a drive will be prepended to newdir, its buffer
**           should be at least MAX_FLEN long.
*/

int PushDir(char *newdir)
{
      char pname[MAX_FLEN];
      char drive[3];
      char *target = &pname[2];
      int  new_drv = 0, ercode = 0;
      static int init = 0;

      if (!init)
            PushDir_stack_ptr = init = -1;
      if (DIR_STACK_SIZE <= ++PushDir_stack_ptr)
      {
            ercode = -1;
            goto ErrEx;
      }
      getcwd(PushDir_stack[PushDir_stack_ptr], MAX_FLEN);
      strupr(PushDir_stack[PushDir_stack_ptr]);
      strncpy(drive, PushDir_stack[PushDir_stack_ptr], 2);
      drive[2] = '\0';
      if (':' == newdir[1])
      {     /* If a drive is specified                                  */
            strupr(newdir);
            strcpy(pname, newdir);
            if (strchr(target, ':'))      /* if filename is illegal     */
                  goto ErrEx;
            if (*drive != *newdir)
            {
                  if (Error_ == chdrv(newdir[0] - 'A'))
                  {     /* If the drive is invalid                    */
                        goto ErrEx;
                  }
                  else  new_drv = 1;
            }
      }
      else
      {     /* If a drive isn't specified                               */
            if (!strchr(strupr(newdir), ':'))
            {     /* If legal filename                                  */
                  strcpy(pname, drive);
                  strcat(pname, newdir);
                  strcpy(newdir, pname);
            }
            else
            {     /* If filename is illegal                             */
                  goto ErrEx;
            }
      }

      if (*target)
      {
```

```
101 |             if (chdir(target))
102 |             {
103 |                   if (1 == new_drv) /* We already changed drives        */
104 |                         chdrv(*drive - 'A');    /* Go home before exit  */
105 |                   goto ErrEx;
106 |             }
107 |       }
108 |       return (new_drv + 1);
109 | ErrEx:
110 |       --PushDir_stack_ptr;
111 |       return (ercode);
112 | }
113 |
114 | /*
115 | **  PopDir()
116 | **
117 | **  Like chdir(), except goes to the drive/directory specified on the
118 | **  top of the PushDir stack.
119 | **
120 | **  Arguments: none
121 | **
122 | **  Returns:  -1 - stack empty
123 | **             0 - error - stack pointer unchanged
124 | **             1 - success, still on same drive
125 | **             2 - success, changed drive
126 | **
127 | **  Side effects: none
128 | **
129 | **  CAUTION: chdir() or chdrv() should not be called between PushDir-
130 | **           PopDir calls.
131 | */
132 |
133 | int PopDir(void)
134 | {
135 |       char I_am_here[MAX_FLEN], target_drv, *target;
136 |       int new_drv = 0;
137 |
138 |       if (0 > PushDir_stack_ptr)
139 |             return -1;
140 |       getcwd(I_am_here, MAX_FLEN);
141 |       target = &PushDir_stack[PushDir_stack_ptr][2];
142 |       target_drv = PushDir_stack[PushDir_stack_ptr][0];
143 |       if (I_am_here[0] != target_drv)
144 |       {
145 |             if (Error_ == chdrv(target_drv - 'A'))
146 |                   return 0;
147 |             new_drv = 1;
148 |       }
149 |       if (!chdir(target))
150 |       {
151 |             --PushDir_stack_ptr;
152 |             return (1 + new_drv);
153 |       }
154 |       else  return 0;
155 | }
156 |
157 | /*
158 | **  isdir()
159 | **
160 | **  Checks to see if a drive and/or path are a valid directory.
161 | **
162 | **  Arguments: 1 - dir, the buffer containing the new directory name
163 | **
164 | **  Returns: Error_ - push/popdir stack overflow
165 | **           False_ - not a valid directory
166 | **           True_  - valid directory
167 | **
168 | **  Side effects: Converts name in dir to upper case and prepends a
169 | **                drive letter.
170 | **
171 | **  CAUTION: Since a drive will be prepended to newdir, it's buffer
172 | **           should be at least MAX_FLEN long.
173 | */
174 |
175 | int isdir(char *dir)
176 | {
177 |       int ercode;
178 |
179 |       if (-1 == (ercode = PushDir(dir)))
180 |             return ercode;
181 |       if (ercode)
182 |             PopDir();
183 |       return TOBOOL(ercode);
184 | }
```

TEXT STATISTICS

```
  5134 characters
   184 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     7 constants [character]
     2 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    DIR_STACK_SIZE          25      29      61
    ErrEx           64      75      80      95     105     109
    Error_          78     145
    I_am_here      135     140     143
    MAX_FLEN        26      29      53      66     135     140
    MSDOS           19
    PopDir         133     182
    PushDir         51     179
    PushDir_stack           29      66      67      68     141     142
    PushDir_stack_ptr       28      60      61      66      67      68     110     138     141     142
          151
    TOBOOL         183
    __MSDOS__       19
    chdir          101     149
    chdrv           78     104     145
    ctype           17
    defined         19+
    dir            175     179
    dos             14
    drive           54      68      69      76      89     104
    ercode          56      63     111     177     179     180     181     183
    getcwd          66     140
    h       14      15      16      17      22
    init            57      59      60
    isdir          175
    new_drv         56      82     103     108     136     147     152
    newdir          51      70      72      73      76      78      87      90      91
    pname           53      55      73      89      90      91
    stdlib          15
    strcat          90
    strchr          74      87
    strcpy          73      89      91
    string          16
    strncpy         68
    strupr          67      72      87
    target          55      74      99     101     135     141     149
    target_drv             135     142     143     145
    unistd          22
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  QuickMenu VidMgr demonstration application.
 *
 *  Written in June 1996 by Andrew Clarke and released to the public domain.
 *
 *  Currently assumes 80x25 video output, but may be rewritten to
 *  recognise other video dimensions.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vidmgr.h"
#include "vioimage.h"

#define maxx  vm_getscreenwidth()
#define maxy  vm_getscreenheight()

#define PROG     "QuickMenu"
#define VERSION  "1.2"

#define MAXITEMS 12

typedef struct
{
    char desc[40];
    char cmd[250];
}
MENUDATA;

static VIOIMAGE sysimage;
static MENUDATA menudata[MAXITEMS];
static int menuitems, curritem, olditem;

int qmenuLoadMenu(char *fnm)
{
    FILE *fp;
    int item, line;

    fp = fopen(fnm, "r");
    if (fp == NULL)
    {
        perror(fnm);
        return 0;
    }

    item = 0;
    line = 1;
    while (item < MAXITEMS && !feof(fp))
    {
        char *p;

        if (fgets(menudata[item].desc, 40, fp) == NULL)
        {
            break;
        }

        p = strrchr(menudata[item].desc, '\n');
        if (p != NULL)
        {
            *p = '\0';
        }

        if (*menudata[item].desc == '\0')
        {
            printf("%s(%d): Error! Expected description but found ''\n", fnm, line);
            return 0;
        }

        line++;
        if (fgets(menudata[item].cmd, 250, fp) == NULL)
        {
            printf("%s(%d): Error! Expected command but reached end of file\n", fnm, line);
            return 0;
        }

        p = strrchr(menudata[item].cmd, '\n');
        if (p != NULL)
        {
            *p = '\0';
        }

        if (*menudata[item].cmd == '\0')
        {
            printf("%s(%d): Error! Expected command but found ''\n", fnm, line);
            return 0;
        }

        line++;
        item++;
    }

    fclose(fp);

    menuitems = item;

    return 1;
}
```

```
101 |
102 | void qmenuRun(void)
103 | {
104 |     int done, ch;
105 |     FILE *ofp;
106 |     char qmtmpfnm[12];
107 |
108 | #if defined(OS2)
109 |     strcpy(qmtmpfnm, "$qmtemp.cmd");
110 | #elif defined(EMX)
111 |     if (_osmode == DOS_MODE)
112 |     {
113 |         strcpy(qmtmpfnm, "$qmtemp.bat");
114 |     }
115 |     else
116 |     {
117 |         strcpy(qmtmpfnm, "$qmtemp.cmd");
118 |     }
119 | #else
120 |     strcpy(qmtmpfnm, "$qmtemp.bat");
121 | #endif
122 |
123 |     vm_attrib(24, (char) (7 + curritem), 56, (char) (7 + curritem), vm_mkcolor(WHITE, BLACK));
124 |
125 |     done = 0;
126 |     while (!done)
127 |     {
128 |         if (curritem != olditem)
129 |         {
130 |             vm_attrib(24, (char) (7 + olditem), 56, (char) (7 + olditem), vm_mkcolor(BLACK, LIGHTGRAY)
    |             );
131 |             vm_attrib(24, (char) (7 + curritem), 56, (char) (7 + curritem), vm_mkcolor(WHITE, BLACK));
132 |         }
133 |
134 |         while (!vm_kbhit())
135 |         {
136 |             /* nada */
137 |         }
138 |
139 |         ch = vm_getch();
140 |         switch (ch)
141 |         {
142 |         case 0x3b00:    /* F1 */
143 |             /* nada */
144 |             break;
145 |
146 |         case 0x4800:    /* up arrow */
147 |             olditem = curritem;
148 |             if (curritem != 0)
149 |             {
150 |                 curritem--;
151 |             }
152 |             break;
153 |
154 |         case 0x5000:    /* down arrow */
155 |             olditem = curritem;
156 |             if (curritem != menuitems - 1)
157 |             {
158 |                 curritem++;
159 |             }
160 |             break;
161 |
162 |         case 0x4700:    /* Home */
163 |             olditem = curritem;
164 |             curritem = 0;
165 |             break;
166 |
167 |         case 0x4f00:    /* End */
168 |             olditem = curritem;
169 |             curritem = menuitems - 1;
170 |             break;
171 |
172 |         case 0x001b:    /* Escape */
173 |         case 0x3d00:    /* F3 */
174 |         case 0x4400:    /* F10 */
175 |         case 0x6b00:    /* Alt+F4 */
176 |         case 0x2d00:    /* Alt+X */
177 |             remove(qmtmpfnm);
178 |             done = 1;
179 |             break;
180 |
181 |         case 0x000d:    /* Enter */
182 |             ofp = fopen(qmtmpfnm, "w");
183 |             fputs("@echo off\n", ofp);
184 |             fputs(menudata[curritem].cmd, ofp);
185 |             fputs("\n", ofp);
186 |             fputs("qm\n", ofp);
187 |             fclose(ofp);
188 |             done = 1;
189 |             break;
190 |
191 |         default:
192 |             break;
193 |         }
194 |     }
195 | }
196 |
197 | void qmenuTerm(void)
198 | {
199 |     vioImageRestore(&sysimage, 1, 1);
```

```
200 |     vioImageTerm(&sysimage);
201 |     vm_gotoxy(vm_startup.xpos, vm_startup.ypos);
202 |     vm_done();
203 | }
204 |
205 | void qmenuInit(void)
206 | {
207 |     int item;
208 |
209 |     printf("\n" PROG " " VERSION "; Written in June 1996 by Andrew Clarke.\n");
210 |     printf("Released to the public domain.\n");
211 |
212 |     vm_init();
213 |     vioImageDefaults(&sysimage);
214 |     vioImageInit(&sysimage, maxx, maxy);
215 |     vioImageSave(&sysimage, 1, 1);
216 |
217 |     if (qmenuLoadMenu("qmenu.mnu") != 1)
218 |     {
219 |         vm_done();
220 |         exit(EXIT_FAILURE);
221 |     }
222 |
223 |     vm_setcursorstyle(CURSORHIDE);
224 |
225 |     vm_attrib(22, 6, 62, 21, vm_mkcolor(LIGHTGRAY, BLACK));
226 |     vm_paintclearbox(20, 5, 60, 20, vm_mkcolor(BLACK, LIGHTGRAY));
227 |     vm_frame(23, 6, 57, 19, vm_mkcolor(BLACK, LIGHTGRAY), vm_frame_double);
228 |
229 |     for (item = 0; item < menuitems; item++)
230 |     {
231 |         vm_puts(25, (char) (7 + item), menudata[item].desc);
232 |     }
233 | }
234 |
235 | int main(void)
236 | {
237 |     qmenuInit();
238 |     qmenuRun();
239 |     qmenuTerm();
240 |     return 0;
241 | }
```

TEXT STATISTICS

5258 characters
241 lines

LEXICAL STATISTICS

15 comments [std-C]
0 comments [C++]
14 preprocessor instructions
6 constants [character]
21 constants [string]
64 constants [numeric]

SYMBOL TABLE

```
BLACK           123     130     131     225     226     227
CURSORHIDE              223
DOS_MODE        111
EMX             110
EXIT_FAILURE            220
FILE             39     105
LIGHTGRAY       130     225     226     227
MAXITEMS         24      34      51
MENUDATA         31      34
NULL             43      55      61      73      80
OS2             108
PROG             21     209
VERSION          22     209
VIOIMAGE         33
WHITE           123     131
_osmode         111
ch              104     139     140
cmd              29      73      79      85     184
curritem         35     123+    128     131+    147     148     150     155     156     158     163
       164      168     169     184
defined         108     110
desc             28      55      60      66     231
done            104     125     126     178     188
exit            220
fclose           95     187
feof             51
fgets            55      73
fnm              37      42      45      68      75      87
fopen            42     182
fp               39      42      43      51      55      73      95
fputs           183     184     185     186
h       12       13      14
item             40      49      51      55      60      66      73      79      85      92      97
       207      229+    231+
line             40      50      68      72      75      87      91
main            235
maxx             18     214
maxy             19     214
menudata         34      55      60      66      73      79      85     184     231
menuitems        35      97     156     169     229
ofp             105     182     183     184     185     186     187
olditem          35     128     130+    147     155     163     168
p       53       60      61      63      79      80      82
perror           45
printf           68      75      87     209     210
qmenuInit       205     237
qmenuLoadMenu            37     217
qmenuRun        102     238
qmenuTerm       197     239
qmtmpfnm        106     109     113     117     120     177     182
remove          177
stdio            12
stdlib           13
strcpy          109     113     117     120
string           14
strrchr          60      79
sysimage         33     199     200     213     214     215
vioImageDefaults        213
vioImageInit            214
vioImageRestore         199
vioImageSave            215
vioImageTerm            200
vm_attrib       123     130     131     225
vm_done         202     219
vm_frame        227
vm_frame_double         227
vm_getch        139
vm_getscreenheight               19
vm_getscreenwidth        18
vm_gotoxy       201
vm_init         212
vm_kbhit        134
vm_mkcolor              123     130     131     225     226     227
vm_paintclearbox        226
vm_puts         231
vm_setcursorstyle       223
vm_startup              201+
xpos            201
ypos            201
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  QUERY.C - Timed query with default for batch files
**
**  Usage: QUERY [Prompt_string] [X] [n]
**         where: X = Default answer, `Y' or `N'
**                n = Seconds to wait before using default answer
**
**  Note:  If any option is used, those preceding it must be specified.
**         For example, to use the time out feature, both the prompt
**         string and default answer must have previously been specified.
**
**  public domain by Bob Stout
*/

#include <stdio.h>
#include <time.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>

main(int argc, char *argv[])
{
      int ch = '\0', def_ch = '\0';
      char *prompt = "(y/n) ";
      clock_t start, limit = (clock_t)0;

      if (1 < argc)
      {
            fputs(argv[1], stderr);
            fputc(' ', stderr);
      }
      if (2 < argc)
      {
            def_ch = toupper(*argv[2]);
            if ('Y' == def_ch)
                  prompt[1] = def_ch;
            else if ('N' == def_ch)
                  prompt[3] = def_ch;
            else  def_ch = '\0';
      }
      fputs(prompt, stderr);
      if (3 < argc)
      {
            start = clock();
            limit = (clock_t)(CLK_TCK * atoi(argv[3]));
      }
      while ('Y' != ch && 'N' != ch)
      {
            while (!kbhit())
            {
                  if (limit && (limit <= (clock() - start)))
                  {
                        ch = def_ch;
                        goto BYE;
                  }
            }
            ch = toupper(getch());
            if ('Y' != ch && 'N' != ch && (1 < argc))
                  ch = def_ch;
      };
BYE:  fputc(ch, stderr);
      fputc('\n', stderr);
      return ('Y' == ch);
}
```

TEXT STATISTICS

```
  1806 characters
    66 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
    12 constants [character]
     1 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

BYE		56	63						
CLK_TCK		47							
argc		23	29	34	44	60			
argv		23	31	36	47				
atoi		47							
ch		25	49+	55	59	60+	61	63	65
clock		46	53						
clock_t		27+	47						
conio		21							
ctype		19							
def_ch		25	36	37	38	39	40	41	55	61
fputc		32	63	64					
fputs		31	43						
getch		59							
h	17	18	19	20	21				
kbhit		51							
limit		27	47	53+					
main		23							
prompt		26	38	40	43				
start		27	46	53					
stderr		31	32	43	63	64			
stdio		17							
stdlib		20							
time		18							
toupper		36	59						

```
/* +++Date last modified: 05-Jul-1997 */

/*
+-----------------------------------------------------+
|               Thunderbird Software                  |
+-----------------------------------------------------+
| Filespec  :  Queue.c                                |
| Date      :  September 29, 1994                     |
| Time      :  10:16AM                                |
| Revision  :  1.0                                    |
+-----------------------------------------------------+
| Programmer:  Scott Andrews                          |
| Address   :  5358 Summit RD SW                      |
| City/State:  Pataskala, Ohio                        |
| Zip       :  43062                                  |
+-----------------------------------------------------+
| Released to the Public Domain                       |
+-----------------------------------------------------+
*/

#include <stdlib.h>

#include "queue.h"

QUEUE *alloc_queue( int size)
{ QUEUE *retval;
  retval = (QUEUE *) malloc( sizeof( QUEUE) + (size_t) size);
  if ( (QUEUE *) 0 != retval)
  {  retval->size = size;
     retval->head = 0;
     retval->tail = 0;
     retval->avail = size;
     retval->buffer = ( (char *) retval) + sizeof( QUEUE);
  }
  return retval;
}

int en_queue( QUEUE *queue, char data)
{ int retval = -1;
  if ( 0 != queue->avail)
  {  *( queue->buffer + queue->head) = data;
     queue->head += 1;
     if ( queue->head == queue->size)
        queue->head = 0;
     queue->avail -= 1;
     retval = queue->avail;
  }
  return retval;
}

int de_queue( QUEUE *queue)
{ int retval = -1;
  if ( queue->avail != queue->size)
  {  retval = *( queue->buffer + queue->tail);
     queue->tail += 1;
     if ( queue->tail == queue->size)
        queue->tail = 0;
     queue->avail += 1;
  }
  return retval;
}

/* End of Queue.c */
```

TEXT STATISTICS

```
  1875 characters
    63 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
    12 constants [numeric]
```

SYMBOL TABLE

```
    QUEUE           25      26      27+     28      33      38      51
    alloc_queue             25
    avail           32      40      45      46      53      58
    buffer          33      41      54
    data            38      41
    de_queue        51
    en_queue        38
    h       21
    head            30      41      42      43      44
    malloc          27
    queue           38      40      41+     42      43+     44      45      46      51      53+     54+
            55      56+     57      58
    retval          26      27      28      29      30      31      32      33+     35      39      46
            48      52      54      60
    size            25      27      29+     32      43      53      56
    size_t          27
    stdlib          21
    tail            31      54      55      56      57
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | +-----------------------------------------------------+
  5 | |                Thunderbird Software                 |
  6 | +-----------------------------------------------------+
  7 | | Filespec  :  QUEUE.H                                |
  8 | | Date      :  August 30, 1994                        |
  9 | | Time      :  5:40 PM                                |
 10 | | Revision  :  0.0                                    |
 11 | +-----------------------------------------------------+
 12 | | Programmer:  Scott Andrews                          |
 13 | | Address   :  5358 Summit RD SW                      |
 14 | | City/State:  Pataskala, Ohio                        |
 15 | | Zip       :  43062                                  |
 16 | +-----------------------------------------------------+
 17 | | Released to the Public Domain                       |
 18 | +-----------------------------------------------------+
 19 | */
 20 |
 21 | #ifndef QUEUE__H
 22 | #define QUEUE__H
 23 |
 24 | /* Needed by Serial.C */
 25 |
 26 | typedef struct
 27 | { int   size;
 28 |   int   head;
 29 |   int   tail;
 30 |   int   avail;
 31 |   char *buffer;
 32 | } QUEUE;
 33 |
 34 | #define queue_empty(queue) (queue)->head == (queue)->tail
 35 | #define queue_avail(queue) (queue)->avail
 36 |
 37 | QUEUE *alloc_queue( int size);
 38 | int   en_queue( QUEUE *queue_ptr, char data);
 39 | int   de_queue( QUEUE *queue_ptr);
 40 |
 41 | /* End of Queue.H */
 42 |
 43 | #endif /* QUEUE__H */
```

TEXT STATISTICS

```
  1334 characters
    43 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    QUEUE           32      37      38      39
    QUEUE__H        21      22
    alloc_queue             37
    avail           30      35
    buffer          31
    data            38
    de_queue        39
    en_queue        38
    head            28      34
    queue           34+     35+
    queue_avail             35
    queue_empty             34
    queue_ptr       38      39
    size            27      37
    tail            29      34
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  RAD2DEG.C - Functions to convert between radians and degrees
  5 | */
  6 |
  7 | #include <math.h>
  8 | #include "snipmath.h"
  9 |
 10 | #undef rad2deg                /* These are macros defined in PI.H */
 11 | #undef deg2rad
 12 |
 13 | double rad2deg(double rad)
 14 | {
 15 |       return (180.0 * rad / (PI));
 16 | }
 17 |
 18 | double deg2rad(double deg)
 19 | {
 20 |       return (PI * deg / 180.0);
 21 | }
 22 |
 23 | #ifdef TEST
 24 |
 25 | #include <stdio.h>
 26 |
 27 | main()
 28 | {
 29 |       double X;
 30 |
 31 |       for (X = 0.0; X <= 360.0; X += 45.0)
 32 |             printf("%3.0f degrees = %.12f radians\n", X, deg2rad(X));
 33 |       puts("");
 34 |       for (X = 0.0; X <= (2 * PI + 1e-6); X += (PI / 6))
 35 |             printf("%.12f radians = %3.0f degrees\n", X, rad2deg(X));
 36 |       return 0;
 37 | }
 38 |
 39 | #endif /* TEST */
```

TEXT STATISTICS

```
   763 characters
    39 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     4 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    PI              15      20      34+
    TEST            23
    X       29      31+     32+     34+     35+
    deg             18      20
    deg2rad         11      18      32
    h        7      25
    main            27
    math             7
    printf          32      35
    puts            33
    rad             13      15
    rad2deg         10      13      35
    stdio           25
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************
 This random number generator originally appeared in "Toward a Universal
 Random Number Generator" by George Marsaglia and Arif Zaman.
 Florida State University Report: FSU-SCRI-87-50 (1987)

 It was later modified by F. James and published in "A Review of Pseudo-
 random Number Generators"

 Converted from FORTRAN to C by Phil Linttell, James F. Hickling
 Management Consultants Ltd, Aug. 14, 1989.

 THIS IS THE BEST KNOWN RANDOM NUMBER GENERATOR AVAILABLE.
       (However, a newly discovered technique can yield
         a period of 10^600. But that is still in the development stage.)

 It passes ALL of the tests for random number generators and has a period
   of 2^144, is completely portable (gives bit identical results on all
   machines with at least 24-bit mantissas in the floating point
   representation).

 The algorithm is a combination of a Fibonacci sequence (with lags of 97
   and 33, and operation "subtraction plus one, modulo one") and an
   "arithmetic sequence" (using subtraction).

 On a Vax 11/780, this random number generator can produce a number in
    13 microseconds.
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define TRUE    1
#define FALSE   0

float u[97], c, cd, cm;
int i97, j97, test;

int rmarin(int ij, int kl);
int ranmar(float rvec[], int len);


int main()
{

        float temp[100];
        int i;
        int ij, kl, len;

        /*These are the seeds needed to produce the test case results*/

        ij = 1802;
        kl = 9373;

        /*Do the initialization*/

        if (1 == rmarin(ij,kl))
                return 1;

        /*Generate 20000 random numbers*/

        len = 100;
        for ( i=0; i<=199 ; i++)
                if (1 == ranmar(temp, len))
                        return 1;

        /*If the random number generator is working properly,
          the next six random numbers should be:

            6533892.0  14220222.0   7275067.0
            6172232.0   8354498.0  10633180.0
        */

        len = 6;
        if (1 == ranmar(temp, len))
                return 1;

        for ( i=0; i<=5; i++)
                printf("%12.1f\n",4096.0*4096.0*temp[i]);

        return 0;
}


/*****************************************************************
 This is the initialization routine for the random number generator RANMAR()
 NOTE: The seed variables can have values between:    0 <= IJ <= 31328
                                                      0 <= KL <= 30081
 The random number sequences created by these two seeds are of sufficient
 length to complete an entire calculation with. For example, if several
 different groups are working on different parts of the same calculation,
 each group could be assigned its own IJ seed. This would leave each group
 with 30000 choices for the second seed. That is to say, this random
 number generator can create 900 million different subsequences -- with
 each subsequence having a length of approximately 10^30.

 Use IJ = 1802 & KL = 9373 to test the random number generator. The
```

```
 subroutine RANMAR should be used to generate 20000 random numbers.
 Then display the next six random numbers generated multiplied by 4096*4096
 If the random number generator is working properly, the random numbers
 should be:
           6533892.0  14220222.0   7275067.0
           6172232.0   8354498.0  10633180.0
*************************************************************************/

int rmarin(int ij, int kl)
{

        float s, t;
        int i, j, k, l, m;
        int ii, jj;

        /* Change FALSE to TRUE in the next statement to test the
           random routine.*/

        test = TRUE;

        if ( ( ij < 0 || ij > 31328 ) ||
                ( kl < 0 || kl > 30081 ) )
        {
                printf ("RMARIN: The first random number seed must have a "
                        "value between 0 and 31328\n");
                printf ("        The second random number seed must have a "
                        "value between 0 and 30081");
                return 1;
        }

        i = (int)fmod(ij/177.0, 177.0) + 2;
        j = (int)fmod(ij      , 177.0) + 2;
        k = (int)fmod(kl/169.0, 178.0) + 1;
        l = (int)fmod(kl      , 169.0);

        for ( ii=0; ii<=96; ii++ )
        {
                s = (float)0.0;
                t = (float)0.5;
                for ( jj=0; jj<=23; jj++ )
                {
                        m = (int)fmod( fmod(i*j,179.0)*k , 179.0 );
                        i = j;
                        j = k;
                        k = m;
                        l = (int)fmod( 53.0*l+1.0 , 169.0 );
                        if ( fmod(l*m,64.0) >= 32)
                                s = s + t;
                        t = (float)(0.5 * t);
                }
                u[ii] = s;
        }

        c  = (float)(  362436.0 / 16777216.0);
        cd = (float)( 7654321.0 / 16777216.0);
        cm = (float)(16777213.0 / 16777216.0);

        i97 = 96;
        j97 = 32;

        test = TRUE;

        return 0;
}

int ranmar(float rvec[], int len)
{
        float uni;
        int ivec;

        if ( !test )
        {
                printf ("RANMAR: Call the initialization routine (RMARIN) "
                        "before calling RANMAR.\n");
                return 1;
        }

        for ( ivec=0; ivec < len; ivec++)
        {
                uni = u[i97] - u[j97];
                if ( uni < 0.0F )
                        uni = uni + 1.0;
                u[i97] = uni;
                i97--;
                if ( i97 < 0 )
                        i97 = 96;
                j97--;
                if ( j97 < 0 )
                        j97 = 96;
                c = c - cd;
                if ( c < 0.0F )
                        c = c + cm;
                uni = uni - c;
                if ( uni < 0.0F )
                        uni = uni + 1.0;
                rvec[ivec] = uni;
        }
        return 0;
}

```

```
201 | /* I use the following procedure in TC to generate seeds:
202 |
203 |   The sow() procedure calculates two seeds for use with the random number
204 |   generator from the system clock.  I decided how to do this myself, and
205 |   I am sure that there must be better ways to select seeds; hopefully,
206 |   however, this is good enough.  The first seed is calculated from the values
207 |   for second, minute, hour, and year-day; weighted with the second most
208 |   significant and year-day least significant.  The second seed weights the
209 |   values in reverse.
210 | */
211 |
212 | void sow( seed1, seed2 )
213 | int *seed1, *seed2;
214 | {
215 |         struct tm *tm_now;
216 |         float s_sig, s_insig, maxs_sig, maxs_insig;
217 |         time_t secs_now;
218 |         int s, m, h, d, s1, s2;
219 |
220 |         time(&secs_now);
221 |         tm_now = localtime(&secs_now);
222 |
223 |         s = tm_now->tm_sec + 1;
224 |         m = tm_now->tm_min + 1;
225 |         h = tm_now->tm_hour + 1;
226 |         d = tm_now->tm_yday + 1;
227 |
228 |         maxs_sig   = (float)(60.0 + 60.0/60.0 + 24.0/60.0/60.0 +
229 |               366.0/24.0/60.0/60.0);
230 |         maxs_insig = (float)(60.0 + 60.0*60.0 + 24.0*60.0*60.0 +
231 |               366.0*24.0*60.0*60.0);
232 |
233 |         s_sig      = (float)(s + m/60.0 + h/60.0/60.0 + d/24.0/60.0/60.0);
234 |         s_insig    = (float)(s + m*60.0 + h*60.0*60.0 + d*24.0*60.0*60.0);
235 |
236 |         s1 = (int)(s_sig   / maxs_sig   * 31328.0);
237 |         s2 = (int)(s_insig / maxs_insig * 30081.0);
238 |
239 |         *seed1 = s1;
240 |         *seed2 = s2;
241 | }
```

TEXT STATISTICS

```
  7795 characters
   241 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     7 constants [string]
   108 constants [numeric]
```

SYMBOL TABLE

```
    FALSE           37
    TRUE            36     119     161
    c       39     154     190+    191     192+    193
    cd              39     155     190
    cm              39     156     192
    d      218     226     233     234
    fmod           131     132     133     134     142+    146     147
    h       31      32      33      34     218     225     233     234
    i       50      66+     81+     82     113     131     142     143
    i97             40     158     180     183     184     185     186
    ii             114     136+    151
    ij              42      51      55      60     109     121+    131     132
    ivec           169     178+    196
    j      113     132     142     143     144
    j97             40     159     180     187     188     189
    jj             114     140+
    k      113     133     142     144     145
    kl              42      51      56      60     109     122+    133     134
    l      113     134     146+    147
    len             43      51      65      67      77      78     166     178
    localtime      221
    m      113     142     145     147     218     224     233     234
    main            46
    math            33
    maxs_insig             216     230     237
    maxs_sig       216     228     236
    printf          82     124     126     173
    ranmar          43      67      78     166
    rmarin          42      60     109
    rvec            43     166     196
    s      112     138     148+    151     218     223     233     234
    s1             218     236     239
    s2             218     237     240
    s_insig        216     234     237
    s_sig          216     233     236
    secs_now       217     220     221
    seed1          212     213     239
    seed2          212     213     240
    sow            212
    stdio           31
    stdlib          32
    t      112     139     148     149+
    temp            49      67      78      82
    test            40     119     161     171
    time            34     220
    time_t         217
    tm             215
    tm_hour        225
    tm_min         224
    tm_now         215     221     223     224     225     226
    tm_sec         223
    tm_yday        226
    u       39     151     180+    183
    uni            168     180     181     182+    183     193+    194     195+    196
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************
          The McGill Super-Duper Random Number Generator
             G. Marsaglia, K. Ananthanarayana, N. Paul

    Incorporating the Ziggurat method of sampling from decreasing
              or symmetric unimodal density functions.
                      G. Marsaglia, W.W. Tsang

                 Rewritten into C by E. Schneider
 *****************************************************************/

static unsigned long mcgn, srgn;

#define MULT 69069L

void rstart (long i1, long i2)
{
      mcgn = (unsigned long)((i1 == 0L) ? 0L : i1 | 1L);
      srgn = (unsigned long)((i2 == 0L) ? 0L : (i2 & 0x7FFL) | 1L);
}

long uni(void)
{
      unsigned long r0, r1;

      r0 = (srgn >> 15);
      r1 = srgn ^ r0;
      r0 = (r1 << 17);
      srgn = r0 ^ r1;
      mcgn = MULT * mcgn;
      r1 = mcgn ^ srgn;
      return (r1 >> 1);
}

long vni(void)
{
      unsigned long r0, r1;

      r0 = (srgn >> 15);
      r1 = srgn ^ r0;
      r0 = (r1 << 17);
      srgn = r0 ^ r1;
      mcgn = MULT * mcgn;
      r1 = mcgn ^ srgn;
      return r1;
}

/*
"Anyone who consider arithmetic means of producing random number is,
 of course, in a state of sin" - John Von Neumann
*/
```

TEXT STATISTICS

```
  1349 characters
    53 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     0 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    MULT            16      32      45
    i1              18      20+
    i2              18      21+
    mcgn            14      20      32+     33      45+     46
    r0              26      28      29      30      31      39      41      42      43      44
    r1              26      29      30      31      33      34      39      42      43      44      46
            47
    rstart          18
    srgn            14      21      28      29      31      33      41      42      44      46
    uni             24
    vni             37
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  RDXCNVRT.C - Convert between number bases
**
**  public domain demo by Bob Stout
**  uses LTOA.C, also in SNIPPETS
*/

#include <stdlib.h>
#ifdef TEST
 #include <stdio.h>
#endif

char *ltoa(long num, char *buf, int base);

/*
**  Calling parameters: 1 - Number string to be converted
**                      2 - Buffer for the  converted output
**                      3 - Radix (base) of the input
**                      4 - Radix of the output
**
**  Returns: Pointer to converted output
*/

char *radix_convert(const char *in, char *out, int rin, int rout)
{
      long n;
      char *dummy;

      n = strtol(in, &dummy, rin);
      return ltoa(n, out, rout);
}

#ifdef TEST

int main(int argc, char *argv[])
{
      int rin, rout;
      char buf[40];

      if (4 > argc)
      {
            puts("Usage: RDXCNVRT <number> <base_in> <base_out>");
            return(-1);
      }
      rin  = atoi(argv[2]);
      rout = atoi(argv[3]);
      printf("%s (base %d) = %s (base %d)\n", argv[1], rin,
            radix_convert((const char *)argv[1], buf, rin, rout), rout);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1214 characters
    54 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     2 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

TEST		11	35			
argc		37	42			
argv		37	47	48	49	50
atoi		47	48			
base		15				
buf		15	40	50		
dummy		29	31			
h	10	12				
in		26	31			
ltoa		15	32			
main		37				
n	28	31	32			
num		15				
out		26	32			
printf		49				
puts		44				
radix_convert			26	50		
rin		26	31	39	47	49 50
rout		26	32	39	48	50+
stdio		12				
stdlib		10				
strtol		31				

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  REDIRECT.C - Posix-compliant utilities to handle redirection
**
**  written by Benno Sauer, Vienna, 1991
**  released into public domain
*/

#include <stdio.h>
#include <fcntl.h>
#include <sys\stat.h>

#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif
#include "dirport.h"
#include "snpdosys.h"

/*
**  Use these predefined structures for the 3 primary streams.
**  Define others (e.g. stdprn under DOS) as required.
*/

REDIR_STRUCT redir_stdin, redir_stdout, redir_stderr;



void  start_redirect ( REDIR_STRUCT *s )
{
      if (s->which == fileno(stdin))
            s->what = open ( s->path, O_RDWR, S_IREAD );
      else  s->what = open ( s->path, O_CREAT | O_RDWR, S_IREAD | S_IWRITE );

      s->oldhandle = dup ( s->which );

      dup2  ( s->what, s->which );
      close ( s->what );
      s->flag = 1;
}

void  stop_redirect ( REDIR_STRUCT *s )
{
      if ( s->flag )
      {
            dup2 ( s->oldhandle, s->which );
            close ( s->oldhandle );
            s->flag = 0;
      }
}

#ifdef TEST

#include <string.h>

main()
{
      char line[132];
      int i;

      strcpy(redir_stdin.path, "redirect.c");
      redir_stdin.which = fileno(stdin);

      strcpy(redir_stdout.path, "x-file");
      redir_stdout.which = fileno(stdout);

      strcpy(redir_stderr.path, "file.x");
      redir_stderr.which = fileno(stderr);

      start_redirect(&redir_stdin);
      start_redirect(&redir_stdout);
      start_redirect(&redir_stderr);

      for (i = 1; !feof(stdin); ++i)
      {
            if (fgets(line, 132, stdin))
            {
                  fputs(line, stdout);
                  fprintf(stderr, "Read line #%d\n", i);
            }
      }
      fflush(stdout);
      fflush(stderr);

      stop_redirect(&redir_stdin);
      stop_redirect(&redir_stdout);
      stop_redirect(&redir_stderr);

      fputs("All done!\n", stdout);
      fflush(stdout);
      fputs("Hit Enter to exit\n", stderr);
      getchar();

      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

2178 characters
99 lines

LEXICAL STATISTICS

4 comments [std-C]
0 comments [C++]
13 preprocessor instructions
0 constants [character]
9 constants [string]
6 constants [numeric]

SYMBOL TABLE

MSDOS		14										
O_CREAT		35										
O_RDWR		34	35									
REDIR_STRUCT			27	31	44							
S_IREAD		34	35									
S_IWRITE		35										
TEST		54										
__MSDOS__		14										
close		40	49									
defined		14+										
dup		37										
dup2		39	48									
fcntl		11										
feof		76										
fflush		84	85	92								
fgets		78										
fileno		33	64	67	70							
flag		41	46	50								
fprintf		81										
fputs		80	91	93								
getchar		94										
h	10	11	12	17	56							
i	61	76+	81									
line		60	78	80								
main		58										
oldhandle		37	48	49								
open		34	35									
path		34	35	63	66	69						
redir_stderr			27	69	70	74	89					
redir_stdin			27	63	64	72	87					
redir_stdout			27	66	67	73	88					
s	31	33	34+	35+	37+	39+	40	41	44	46	48+	49
	50											
start_redirect			31	72	73	74						
stderr		70	81	85	93							
stdin		33	64	76	78							
stdio		10										
stdout		67	80	84	91	92						
stop_redirect			44	87	88	89						
strcpy		63	66	69								
string		56										
sys\stat		12										
unistd		17										
what		34	35	39	40							
which		33	37	39	48	64	67	70				

```
/* +++Date last modified: 05-Jul-1997 */

/*************************************************************************

   REGIT.C - A very simple registration key generator. Uses simple XOR
   manipulations of a string to create a key.

   It is NOT foolproof, but it will work.

   Donated to the Public Domain by Craig Morrison 12 May 1994, use,
   abuse, fold, spindle or mutilate anyway you see fit.

*************************************************************************/

#include "regkey.h"

/*************************************************************************

    REGIT accepts one argument on the command line; The string you want
    to use to generate a key from. It outputs the generated key in both
    decimal and hexadecimal form. Spaces in the argument should have the
    '_' character used in their place, they get translated below.

*************************************************************************/

int main(int argc, char *argv[])
{
      long keyval = (long)XOR_PRIME;
      long key;
      char *p;
      char buf[128];

      if (argc>1)
      {
            strcpy(buf, argv[1]);
            p = buf;
            while(*p)
            {
                  if (*p=='_')
                        *p = ' ';

                  key = (long) toupper(*p);
                  key ^= (long)XOR_CRYPT;
                  keyval ^= key;
                  p++;
            }
            keyval ^= (long)XOR_POST_CRYPT;
            printf("Key value = %08lX hex, %lu decimal.\n", keyval, keyval);
      }
      return 0;
}
```

```
TEXT STATISTICS

  1599 characters
    51 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     2 constants [character]
     2 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    XOR_CRYPT       43
    XOR_POST_CRYPT          47
    XOR_PRIME       28
    argc            26      33
    argv            26      35
    buf             31      35      36
    key             29      42      43      44
    keyval          28      44      47      48+
    main            26
    p       30      36      37      39      40      42      45
    printf          48
    strcpy          35
    toupper         42
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  REGKEY.H - SNIPPETS header file for REGIT.C and CHKREG.C
   5 | */
   6 |
   7 | #include <stdio.h>
   8 | #include <stdlib.h>
   9 | #include <string.h>
  10 | #include <ctype.h>
  11 |
  12 | /* Choose your own values for these */
  13 |
  14 | #define XOR_PRIME      0xFFFFFFFFL
  15 | #define XOR_CRYPT      0x13579ACEL
  16 | #define XOR_POST_CRYPT 0x2468BDF0L
```

TEXT STATISTICS

```
   350 characters
    16 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     0 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    XOR_CRYPT       15
    XOR_POST_CRYPT          16
    XOR_PRIME       14
    ctype           10
    h         7      8       9      10
    stdio            7
    stdlib           8
    string           9
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 * REMCMMNT.C
 * Remove comments from C or C++ source
 *
 * ver 1.1, 14 Jun 1995
 *   - changed multiline C++ comment handling
 *
 * ver 1.2, 8 Nov 1995
 *   - bug: if file was ended with // comment with no CR/LF,
 *          program jammed in an endless loop.
 *
 * ver 1.3, 21 Nov 1995
 *   - bug: did not handle empty strings ""
 *   - bug: did not handle escape character \" inside strings
 *   - bug: did not catch an end of comment that ended with two or more '*'s
 *
 * Public domain by:
 *   Jari Laaksonen
 *   Arkkitehdinkatu 30 A 2
 *   FIN-33720 Tampere
 *   FINLAND
 *
 *   Fidonet : 2:221/360.20
 *   Internet: jla@to.icl.fi
 */

#include <stdio.h>

int main (int argc, char **argv)
{
  int  Char,
       Char2,
       cpp_comment   = 0,
       c_comment     = 0,
       in_string     = 0,
       cpp_multiline = 0;
  char CannotOpen[] = "Cannot open %s\n\n";
  FILE *InFile, *OutFile = stdout;

  if (argc < 2)
  {
    fprintf (stderr, "USAGE: REMCMMNT InFile [OutFile]\n");
    return 1;
  }
  if ((InFile = fopen (argv[1], "r")) == NULL)
  {
    fprintf (stderr, CannotOpen, argv[1]);
    return 2;
  }

  if (argc == 3)
  {
    if ((OutFile = fopen (argv[2], "w")) == NULL)
    {
      fprintf (stderr, CannotOpen, argv[2]);
      OutFile = stdout;  /* if can't open, output goes to stdout instead */
    }
  }

  while ((Char = fgetc (InFile)) != EOF)
  {
    /* do we have excape characters \", \\ etc. inside string ? */
    if (in_string && Char == '\\')
    {
      Char2 = fgetc (InFile);

      fputc (Char,  OutFile);
      fputc (Char2, OutFile);
      continue;
    }

    if (Char == '\"')
    {
      Char2 = fgetc (InFile);

      /* do we have a character constant '\"' or an empty string "" ? */
      if (Char2 != '\'' && Char2 != '\"')
        in_string = ! in_string;    /* if not, we are in a string */

      if (c_comment == 0 && cpp_comment == 0)
      {
        fputc (Char,  OutFile);
        fputc (Char2, OutFile);
        continue;
      }
    }

    if (! in_string)
    {
      if (Char == '/')
      {
        Char2 = fgetc (InFile);     /* check next char */
        if (Char2 == '/')           /* is it start of C++ comment? */
          cpp_comment = 1;
        else if (Char2 == '*')      /* is it start of C comment? */
          c_comment = 1;

        if (c_comment == 0 && cpp_comment == 0)
```

```
101 |         {
102 |           fputc (Char,  OutFile);
103 |           fputc (Char2, OutFile);
104 |           continue;
105 |         }
106 |       }
107 |       else if (Char == '*' && c_comment)
108 |       {
109 |         Char2 = fgetc (InFile);
110 |         if (Char2 == '/')                  /* is it end of C comment? */
111 |         {
112 |           c_comment = 0;
113 |           Char = fgetc (InFile);
114 |         }
115 |       }
116 |
117 |       if (c_comment || cpp_comment) /* are we inside C or C++ comment? */
118 |       {
119 |         Char = '\n'; /* print newline after comment line is processed */
120 |         while ((Char2 = fgetc (InFile)) != '\n') /* rest of the line */
121 |         {
122 |           if (Char2 == EOF)
123 |             break;
124 |
125 |           if (cpp_multiline)
126 |           {
127 |             if (Char2 != ' ' && Char2 != '\t') /* if not white space => */
128 |               cpp_multiline = 0;             /* ...not C++ multiline comment */
129 |           }
130 |
131 |           if (Char2 == '\\' && cpp_comment)
132 |             cpp_multiline = 1;
133 |
134 |           if (Char2 == '*' && c_comment)
135 |           {
136 |             Char2 = fgetc (InFile); /* check next char */
137 |             if (Char2 == '/')       /* is it end of C comment? */
138 |             {
139 |               c_comment = 0;
140 |               Char = fgetc (InFile);
141 |               break;
142 |             }
143 |             else
144 |               ungetc (Char2, InFile);        /* put it back to stream */
145 |             if (Char2 == '\n')
146 |               break;
147 |           }
148 |         }
149 |         if (cpp_comment && cpp_multiline == 0)
150 |           cpp_comment = 0;
151 |         if (c_comment || cpp_comment)
152 |           fputc (Char,  OutFile);
153 |
154 |         /* clear flag for the next round. if it is still clear after
155 |            next C++ comment line is processed, multiline C++ comment
156 |            is ended.
157 |         */
158 |         cpp_multiline = 0;
159 |       }
160 |
161 |     }
162 |     if (c_comment == 0 && cpp_comment == 0)
163 |       fputc (Char,  OutFile);
164 |
165 |   } /* while end */
166 |
167 |   if (argc == 3)
168 |     fclose (OutFile);
169 |   fclose (InFile);
170 |
171 |   fflush (stdout);
172 |   fprintf (stderr, "\nOK!\n");
173 |
174 |   return 0;
175 | }
```

TEXT STATISTICS

```
  4530 characters
   175 lines
```

LEXICAL STATISTICS

```
    20 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
    17 constants [character]
     5 constants [string]
    29 constants [numeric]
```

SYMBOL TABLE

```
    CannotOpen              39      49      57
    Char             33     62      65      69      74      84      92     102     107     113     119
           140      152     163
    Char2            34     67      70      76     79+      85      94      95      97     103     109
           110      120     122    127+     131     134     136     137     144     145
    EOF              62     122
    FILE             40
    InFile           40      47      62      67      76      94     109     113     120     136     140
           144      169
    NULL             47      55
    OutFile          40      55      58      69      70      84      85     102     103     152     163
           168
    argc             31      42      53     167
    argv             31      47      49      55      57
    c_comment        36      82      98     100     107     112     117     134     139     151     162
    cpp_comment              35      82      96     100     117     131     149     150     151     162
    cpp_multiline            38     125     128     132     149     158
    fclose          168     169
    fflush          171
    fgetc            62      67      76      94     109     113     120     136     140
    fopen            47      55
    fprintf          44      49      57     172
    fputc            69      70      84      85     102     103     152     163
    h       29
    in_string        37      65     80+      90
    main             31
    stderr           44      49      57     172
    stdio            29
    stdout           40      58     171
    ungetc          144
```

```
/* +++Date last modified: 05-Jul-1997 */

/* remtab.c 12-4-91 Robert Mashlan, Public Domain
   modified 28 mar 93 by Bob Stout

   Filter for removing tabs.  All tabs in the input will be replaced
   with spaces.  This filter takes one optional command line
   parameter, which specifies the number spaces to replace for a tab.
   If no size is specifies, it defaults to 8.

   example usage:

   remtab 6 < tabbed.c > untabbed.c

*/

#include <stdio.h>
#include <stdlib.h>

#define BUFSIZE 4096


int main(int argc, char **argv )
{
      int tabsize = 8;

      if (argc > 1)                /* look for command line parameter */
      {
            if (0 == (tabsize = atoi(argv[1])))
                  tabsize = 8;
      }

      while (1)
      {
            char buf[BUFSIZE];
            int nr, i, j, pos = 0;

            nr = fread(buf, 1, sizeof(buf), stdin);
            for (i = 0; i < nr; i++)
            {
                  switch (buf[i])
                  {
                  case '\t':              /* replace tabs with spaces   */
                        for(j = pos % tabsize; j < tabsize; ++j)
                        {
                              putchar(' ');
                              ++pos;
                        }
                        break;

                  case '\n':              /* start a new line           */
                        pos = -1;         /* this will become 0 when... */

                        /* ...we fall through to...   */

                  default:
                        putchar(buf[i]);/* send character through unchanged */
                        ++pos;
                  }
            }
            if (nr < sizeof(buf))
                  break;
      }
      return 0;
}
```

TEXT STATISTICS

```
  1810 characters
    65 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     3 constants [character]
     0 constants [string]
    12 constants [numeric]
```

SYMBOL TABLE

BUFSIZE		20	35			
argc		23	27			
argv		23	29			
atoi		29				
buf		35	38+	41	57	61
fread		38				
h	17	18				
i	36	39+	41	57		
j	36	44+				
main		23				
nr		36	38	39	61	
pos		36	44	47	52	58
putchar		46	57			
stdin		38				
stdio		17				
stdlib		18				
tabsize		25	29	30	44+	

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)repstr
* @(#)- Replace a pattern in a large amount of data using a copy function
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/

/*
#define     TEST
*/
/*
#define     DEBUG
*/

#include    <stdio.h>
#include    <string.h>                          /* memset, strlen, memmove */
#include    "bacstd.h"                          /* filesize, strnsub       */

#if   defined DEBUG
#     define      BUFFERSIZE  80
#     define      BLOKSIZE          10
#else
#     define      BUFFERSIZE  2000
#     define      BLOKSIZE           250
#endif

#if   defined     TEST
#     define      MAXFIELDS   20
#endif

/*
 /-------------------------------------\
|        REPSTR                        |-----------------------------------|
|\-------------------------------------/
|
| Replace a pattern in a large amount of data using a copy function
|
|
|-------------------------------------------------------------------------|
| CALL:
|    repstr( fpInput, fpOutput, pszCharTable ) ;
|
|
| HEADER:
|    stdio.h
|    string.h
|    bacstd.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    %
|
| PROTOTYPE:
|    int _CfnTYPE repstr(FILE *fpIn, FILE *fpOut, char *PatternTable[][2]) ;
|
| RETURN VALUE:
|    int           No of replacements
|
| MODULE:
|    repstr.c
|-------------------------------------------------------------------------|
|
|
|-------------------------------------------------------------------------|
|1993-06-19/Erik Bachmann
\-------------------------------------------------------------------------|*/

int _CfnTYPE repstr(FILE *fpIn, FILE *fpOut, char *PatternTable[][2])
{
      char  szBuffer[BUFFERSIZE+1] ;
      char  szTmpBuffer[BUFFERSIZE+1] ;

      long  lFileLength ;
      long  lFilePos = 0 ;

      int   i = 0 ;
      int   iStrLength ;

      /*--------------------------------------*/

      memset(szBuffer, '\0', BUFFERSIZE) ;

      iStrLength = fread(szBuffer, BLOKSIZE, 1, fpIn); /* Read first block */

      lFilePos = BLOKSIZE ;

      iStrLength = fread(&szBuffer[BLOKSIZE], BLOKSIZE, 1, fpIn) ;

      lFilePos += BLOKSIZE ;

      while( 0 != iStrLength )
      {     /* WHILE able to read */
            i = 0 ;
```

```
101 |
102 |             lFilePos += BLOKSIZE ;
103 |
104 |             while ( NULL != PatternTable[i][0][0] )
105 |             {     /* For all patterns */
106 |                   while (0 != strnsub(szBuffer, PatternTable[i][0],
107 |                                       PatternTable[i][1], BUFFERSIZE))
108 |                   {
109 |                         ;     /* If found replace */
110 |                   }
111 |                   i++ ;
112 |             }     /* Replace pattern with replacement */
113 |
114 |             fwrite(&szBuffer, strlen(szBuffer) - BLOKSIZE, 1, fpOut) ;
115 |
116 |             /* Write szBuffer execept last block */
117 |
118 |             fflush(fpOut) ;
119 |
120 |             memmove(&szBuffer, &szBuffer[strlen(szBuffer) - BLOKSIZE],
121 |                     BLOKSIZE+1) ;
122 |
123 |             memset(&szBuffer[BLOKSIZE], '\0', BLOKSIZE) ;
124 |
125 |             iStrLength = fread(&szBuffer[BLOKSIZE], BLOKSIZE, 1, fpIn) ;
126 |
127 |             szBuffer[BLOKSIZE + BLOKSIZE] = '\0' ;
128 |
129 |       }
130 |
131 |
132 |       i = 0 ;
133 |
134 |       while (PatternTable[i][0][0] != NULL)
135 |       {     /* For all patterns */
136 |             while (0 != strnsub(szBuffer, PatternTable[i][0],
137 |                                 PatternTable[i][1], BUFFERSIZE))
138 |             {
139 |                   ;
140 |             }
141 |             i++ ;
142 |       }
143 |
144 |       /* Replace pattern with replacement */
145 |
146 |       fwrite(&szBuffer, strlen(szBuffer), 1, fpOut) ;
147 |
148 |       /* Write buffer */
149 |
150 |       lFilePos += strlen(&szBuffer[BLOKSIZE]) ;
151 |
152 |       return(0) ;
153 | }
154 |
155 |
156 | #if defined TEST
157 |
158 | int main()
159 | {
160 |       char        *pszTegnTabel[MAXFIELDS][2] = {
161 |                         {"XXX", "YYYY"},
162 |                         {"ZZZ", "A"},
163 |                         {"",""}
164 |                         } ;
165 |
166 |       FILE        *fp1,
167 |                         *fp2 ;
168 |
169 |       /*--------------------------------*/
170 |
171 |       fp1 = fopen("test1", "rt") ;
172 |       fp2 = fopen("test2", "wt") ;
173 |
174 |       if ( (fp1 != NULL) && (fp2 != NULL) )
175 |
176 |             repstr(fp1, fp2, pszTegnTabel) ;
177 |
178 |       fclose(fp1) ;
179 |       fclose(fp2) ;
180 |
181 |       return(0) ;
182 | }
183 |
184 | #endif
```

TEXT STATISTICS

```
  4591 characters
   184 lines
```

LEXICAL STATISTICS

```
    18 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     3 constants [character]
    11 constants [string]
    32 constants [numeric]
```

SYMBOL TABLE

```
    BLOKSIZE        26      29      90      92      94+     96      102     114     120     121     123+
           125+    127+    150
    BUFFERSIZE              25      28      77      78      88      107     137
    DEBUG           24
    FILE            75+     166
    MAXFIELDS       33      160
    NULL            104     134     174+
    PatternTable            75      104     106     107     134     136     137
    TEST            32      156
    _CfnTYPE        75
    define          25      26      28      29      33
    defined         24      32      156
    fclose          178     179
    fflush          118
    fopen           171     172
    fp1             166     171     174     176     178
    fp2             167     172     174     176     179
    fpIn             75      90      94      125
    fpOut            75      114     118     146
    fread            90      94      125
    fwrite          114     146
    h       20      21
    i       83      100     104     106     107     111     132     134     136     137     141
    iStrLength              84      90      94      98      125
    lFileLength             80
    lFilePos         81      92      96      102     150
    main            158
    memmove         120
    memset           88     123
    pszTegnTabel            160     176
    repstr           75     176
    stdio            20
    string           21
    strlen          114     120     146     150
    strnsub         106     136
    szBuffer         77      88      90      94      106     114+    120+    123     125     127     136
           146+    150
    szTmpBuffer              78
```

```
/* +++Date last modified: 05-Jul-1997 */

#define X N(a,O(h,W(f,M(c,g))),O(i,W(f,M(d,g))))
#define A(x) r(D(x,1); O(x,-9); D(x,O(x,1)))
#define L(x) toupper(getchar())-x
#define R Z,Z,0,0
#define S 0,9,6,6,6,6,6,6,9,0
#define q D(h,0); A(f)A(g)
#define T 0,6,1,2,2,2,2,1,6,0
#define U 0,6,2,3,3,3,3,2,6,0
#define C(x) ((x<1)||(x>8))
#define F(x,y) printf(x,y);
#define N(x,y,z) *O(*O(x,y),z)
#define y(a,b,c) a[b][c]
#define O(x,y) ((x)+(y))
#define u(x) (O(0,-(x)))
#define W(x,y) ((x)*(y))
#define G(x) printf(x);
#define D(x,y) (x=(y))
#define P (rand()%6)
#define Y D(e,O(e,1))
#define s D(f,O(f,Q))
#define H(x) return x
#define B(x) while(x)
#define M(x,y) *O(x,y)
#define z(a,b) a[b]
#define E(x) if(x)
#define I main(){
#define Z 0,0,0,0
#define t G("\n")
#define V(x) (!x)
#define v h,i,j,k
#define w e,f,g
#define J int
#define Q u(1)
#define p "%c"
#define o 'A'
#define r for
#define n 60
#define K do

J y(a,10,10)={R,R,R,R,Z,1,Q,Z,Z,Q,1,R,R,R,R,Z},y(b,10,10)={R,S,T,
U       ,       U       ,       U       ,       U       ,       T
,       S       ,       R       }       ,       z       (       c
,       9       )       =       {       Q       ,       Q       ,
Q,0,0,1,1,1,0},z(d,9)={Q,0,1,Q,1,Q,0,1,0};I J w,v;l();r(D(e,0);O(
e       ,       -       n       )       ;       Y       )       {
q       D       (       h       ,       O       (       h       ,
m       (       f       ,       g       ,       1       ,       1
)));E(h){K{G("\n?")K{D(f,L(O('A',Q)));}B(C(f));K{D(g,L('0'));}B(C
(       g       )       )       ;       }       B       (       V
(       m       (       f       ,       g       ,       1       ,
Q       )       )       )       ;       l       (       )       ;
}q E((D(k,O(m(f,g,Q,1),P)))>h){D(h,k);D(i,f);D(j,g);}E(h&&m(i,j,Q
,       Q       )       ) /***/ l /***/ (       )       ;       }
}       J       m       (/*   */v/*****/)       J       v       ;
{       J       w       ; /***/ E /***/ (       N       (       a
,h,i))H(0);E(O(k,Q))D(N(a,h,i),j);D(e,N(b,h,i));r(D(g,1);O(g,-9);
D       (       g       , /***/ O /***/ (       g       ,       1
)       )       )       {/*****/D/*   */(       g       ,       O
(       g       ,       Q /***/ ) /***/ )       ;       E       (
V(O(N(a,O(h,M(c,g)),O(i,M(d,g))),j))){r(D(f,1);V(O(X,j));D(f,O(f,
1       )       )       )       ;       E       (       V       (
O       (       N       (       a       ,       O       (       h
,       W       (       f       ,       M       (       c       ,
g))),O(i,W(f,M(d,g)))),u(j)))r(s;f;s){X=X*k;D(e,O(e,N(b,O(h,W(f,
M       (       c       ,       g       )       )       )       ,
O       (       i       ,       W       (       f       ,       M
(       d       ,       g       )       )       )       )       )
);}}D(g,O(g,1));}E(V(O(e,u(N(b,h,i))))){D(N(a,h,i),0);H(0);}H(e);
}       l       (       )       {       J       f       ,       g
;       t       A       (       g       )       {       F       (
p       ,       O       (       O       (       o       ,       g
),Q))A(f)F("%c:",M("X O<",O(N(a,g,f),1)))t}G(" ")A(f)F("%d ",f)t}
```

TEXT STATISTICS

3167 characters
74 lines

LEXICAL STATISTICS

13 comments [std-C]
0 comments [C++]
38 preprocessor instructions
3 constants [character]
7 constants [string]
82 constants [numeric]

SYMBOL TABLE

A	4	8+	72	74+								
B	24	50+	51									
C	11	50+										
D	4+	8	19	21	22	46	48	50+	54+	58+	59	60
	62+	66	70+									
E	27	50	54+	57	58	61	63	70				
F	12	72	74+									
G	18	30	50	74								
H	23	58	70+									
I	28	46										
J	34	42	46	56+	57	71						
K	40	50+										
L	5	50+										
M	3+	25	62+	65	66	67	68	74				
N	3	13	57	58+	62	64	66	70+	74			
O	3+	4+	13+	15	16	21	22	25	46	48	50	54
	58+	59	60	62+	64+	66+	68	70+	73+	74		
P	20	54										
Q	22	35	42+	45+	46+	50	53	54+	55	58	61	74
R	6	42+	44									
S	7	42	44									
T	9	42	43									
U	10	43+										
V	31	51	62+	63	70							
W	3+	17	65	66+	68							
X	3	62	66+									
Y	21	47										
Z	6+	29	42+									
a	3	14+	26+	42	57	58	62	64	70	74		
b	14+	26+	42	58	66	70						
c	3	14+	44	62	65	67						
d	3	46	62	66	69							
e	21+	33	46	47	58	66+	70+					
f	3+	8	22+	33	49	50+	52	54+	62+	65	66+	68
	71	74+										
g	3+	8	33	49	50	51	52	54+	58+	59+	60	61
	62+	66+	67	69	70+	71	72	73	74			
getchar		5										
h	3	8	32	48+	50	54+	58+	62	64	66	70+	
i	3	32	54+	58+	62	66	68	70+				
j	32	54+	58	62+	66							
k	32	54+	58	66								
l	46	53	55	71								
m	49	52	54+	56								
main		28										
n	39	47										
o	37	73										
p	36	73										
printf		12	18									
q	8	48	54									
r	4	38	46	58	62	66						
rand		20										
s	22	66+										
t	30	72	74+									
toupper		5										
u	16	35	66	70								
v	32	46	56+									
w	33	46	57									
x	4+	5+	11+	12+	13+	15+	16+	17+	18+	19+	23+	24+
	25+	27+	31+									
y	12+	13+	14	15+	17+	19+	25+	42+				
z	13+	26	44	46								

```
/* +++Date last modified: 05-Jul-1997 */

/*
** quicksort.c -- quicksort integer array
**
** public domain by Raymond Gardner     12/91
*/

#include "snipsort.h"

static void swap(int *a, int *b)
{
      register int t;

      t = *a;
      *a = *b;
      *b = t;
}

void quicksort(int v[], unsigned n)
{
      unsigned i, j, ln, rn;

      while (n > 1)
      {
            swap(&v[0], &v[n/2]);
            for (i = 0, j = n; ; )
            {
                  do
                        --j;
                  while (v[j] > v[0]);
                  do
                        ++i;
                  while (i < j && v[i] < v[0]);
                  if (i >= j)
                        break;
                  swap(&v[i], &v[j]);
            }
            swap(&v[j], &v[0]);
            ln = j;
            rn = n - ++j;
            if (ln < rn)
            {
                  quicksort(v, ln);
                  v += j;
                  n = rn;
            }
            else
            {
                  quicksort(v + j, rn);
                  n = ln;
            }
      }
}
```

TEXT STATISTICS

```
  1140 characters
    54 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    a       11      15      16
    b       11      16      17
    i       22      27      33      34+     35      37
    j       22      27      30      31      34      35      37      39      40      41      45      50
    ln              22      40      42      44      51
    n       20      24      26      27      41      46      51
    quicksort       20      44      50
    rn              22      41      42      46      50
    swap            11      26      37      39
    t       13      15      17
    v       20      26+     31+     34+     37+     39+     44      45      50
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** isort()  --  insertion sort
  5 | **
  6 | ** Raymond Gardner   public domain   2/93
  7 | **
  8 | ** qsort() compatible, but uses insertion sort algorithm.
  9 | */
 10 |
 11 | #include <stddef.h>            /* for size_t definition */
 12 | #include "snipsort.h"
 13 |
 14 | void isort(void *base, size_t nmemb, size_t size,
 15 |            int (*comp)(const void *, const void *))
 16 | {
 17 |       char *i, *j, *lim;
 18 |
 19 |       lim = (char *)base + nmemb * size;    /* pointer past end of array */
 20 |       for ( j = (char *)base, i = j+size; i < lim; j = i, i += size )
 21 |       {
 22 |             for ( ; comp((void *)j, (void *)(j+size)) > 0; j -= size )
 23 |             {
 24 |                   char *a, *b;
 25 |                   char tmp;
 26 |                   size_t k = size;
 27 |
 28 |                   a = j;
 29 |                   b = a + size;
 30 |                   do
 31 |                   {
 32 |                         tmp = *a;
 33 |                         *a++ = *b;
 34 |                         *b++ = tmp;
 35 |                   } while ( --k );
 36 |                   if ( j == (char *)base )
 37 |                         break;
 38 |             }
 39 |       }
 40 | }
```

TEXT STATISTICS

```
  1110 characters
    40 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    a       24      28      29      32      33
    b       24      29      33      34
    base            14      19      20      36
    comp            15      22
    h       11
    i       17      20+
    isort           14
    j       17      20+     22+     28      36
    k       26      35
    lim             17      19      20
    nmemb           14      19
    size            14      19      20+     22+     26      29
    size_t          14+     26
    stddef          11
    tmp             25      32      34
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************/
/* qsort.c  --  Non-Recursive ANSI Quicksort function            */
/*                                                               */
/* Public domain by Raymond Gardner, Englewood CO  February 1991 */
/*                                                               */
/* Usage:                                                        */
/*     qsort(base, nbr_elements, width_bytes, compare_function); */
/*        void *base;                                            */
/*        size_t nbr_elements, width_bytes;                      */
/*        int (*compare_function)(const void *, const void *);   */
/*                                                               */
/* Sorts an array starting at base, of length nbr_elements, each */
/* element of size width_bytes, ordered via compare_function,    */
/* which is called as  (*compare_function)(ptr_to_element1,      */
/* ptr_to_element2) and returns < 0 if element1 < element2,      */
/* 0 if element1 = element2, > 0 if element1 > element2.         */
/* Most refinements are due to R. Sedgewick. See "Implementing   */
/* Quicksort Programs", Comm. ACM, Oct. 1978, and Corrigendum,   */
/* Comm. ACM, June 1979.                                         */
/*****************************************************************/

#include <stddef.h>                     /* for size_t definition  */
#include "snipsort.h"

/* prototypes */
void qsort(void *, size_t, size_t,
                        int (*)(const void *, const void *));
void swap_chars(char *, char *, size_t);

/*
** Compile with -DSWAP_INTS if your machine can access an int at an
** arbitrary location with reasonable efficiency.  (Some machines
** cannot access an int at an odd address at all, so be careful.)
*/

#ifdef   SWAP_INTS
 void swap_ints(char *, char *, size_t);
 #define  SWAP(a, b)  (swap_func((char *)(a), (char *)(b), width))
#else
 #define  SWAP(a, b)  (swap_chars((char *)(a), (char *)(b), size))
#endif

#define  COMP(a, b)  ((*comp)((void *)(a), (void *)(b)))

#define  T           7    /* subfiles of T or fewer elements will */
                          /* be sorted by a simple insertion sort */
                          /* Note!  T must be at least 3          */

void qsort(void *basep, size_t nelems, size_t size,
                           int (*comp)(const void *, const void *))
{
   char *stack[40], **sp;       /* stack and stack pointer        */
   char *i, *j, *limit;         /* scan and limit pointers        */
   size_t thresh;               /* size of T elements in bytes    */
   char *base;                  /* base pointer as char *         */

#ifdef   SWAP_INTS
   size_t width;                /* width of array element         */
   void (*swap_func)(char *, char *, size_t); /* swap func pointer*/

   width = size;                /* save size for swap routine     */
   swap_func = swap_chars;      /* choose swap function           */
   if ( size % sizeof(int) == 0 ) {   /* size is multiple of ints */
      width /= sizeof(int);           /* set width in ints        */
      swap_func = swap_ints;          /* use int swap function    */
   }
#endif

   base = (char *)basep;        /* set up char * base pointer     */
   thresh = T * size;           /* init threshold                 */
   sp = stack;                  /* init stack pointer             */
   limit = base + nelems * size;/* pointer past end of array      */
   for ( ;; ) {                 /* repeat until break...          */
      if ( limit - base > thresh ) {  /* if more than T elements  */
                                      /*   swap base with middle  */
         SWAP((((limit-base)/size)/2)*size+base, base);
         i = base + size;             /* i scans left to right    */
         j = limit - size;            /* j scans right to left    */
         if ( COMP(i, j) > 0 )        /* Sedgewick's              */
            SWAP(i, j);               /*    three-element sort    */
         if ( COMP(base, j) > 0 )     /*        sets things up    */
            SWAP(base, j);            /*            so that       */
         if ( COMP(i, base) > 0 )     /*      *i <= *base <= *j   */
            SWAP(i, base);            /* *base is pivot element   */
         for ( ;; ) {                 /* loop until break         */
            do                        /* move i right             */
               i += size;             /*        until *i >= pivot */
            while ( COMP(i, base) < 0 );
            do                        /* move j left              */
               j -= size;             /*        until *j <= pivot */
            while ( COMP(j, base) > 0 );
            if ( i > j )              /* if pointers crossed      */
               break;                 /*     break loop           */
            SWAP(i, j);       /* else swap elements, keep scanning*/
         }
         SWAP(base, j);         /* move pivot into correct place  */
         if ( j - base > limit - i ) {  /* if left subfile larger */
            sp[0] = base;               /* stack left subfile base  */
```

```
101 |             sp[1] = j;                /*    and limit             */
102 |             base = i;                 /* sort the right subfile   */
103 |          } else {                     /* else right subfile larger*/
104 |             sp[0] = i;                /* stack right subfile base */
105 |             sp[1] = limit;            /*    and limit             */
106 |             limit = j;                /* sort the left subfile    */
107 |          }
108 |          sp += 2;                     /* increment stack pointer  */
109 |       } else {      /* else subfile is small, use insertion sort  */
110 |          for ( j = base, i = j+size; i < limit; j = i, i += size )
111 |             for ( ; COMP(j, j+size) > 0; j -= size ) {
112 |                SWAP(j, j+size);
113 |                if ( j == base )
114 |                   break;
115 |             }
116 |          if ( sp != stack ) {         /* if any entries on stack  */
117 |             sp -= 2;                  /* pop the base and limit   */
118 |             base = sp[0];
119 |             limit = sp[1];
120 |          } else                       /* else stack empty, done   */
121 |             break;
122 |       }
123 |    }
124 | }
125 |
126 | /*
127 | **  swap nbytes between a and b
128 | */
129 |
130 | static void swap_chars(char *a, char *b, size_t nbytes)
131 | {
132 |    char tmp;
133 |    do {
134 |       tmp = *a; *a++ = *b; *b++ = tmp;
135 |    } while ( --nbytes );
136 | }
137 |
138 | #ifdef   SWAP_INTS
139 |
140 | /*
141 | **  swap nints between a and b
142 | */
143 |
144 | static void swap_ints(char *ap, char *bp, size_t nints)
145 | {
146 |    int *a = (int *)ap, *b = (int *)bp;
147 |    int tmp;
148 |    do {
149 |       tmp = *a; *a++ = *b; *b++ = tmp;
150 |    } while ( --nints );
151 | }
152 |
153 | #endif
```

TEXT STATISTICS

```
  6883 characters
   153 lines
```

LEXICAL STATISTICS

```
    77 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     1 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    COMP            45      81      83      85      90      93     111
    SWAP            40      42      78      82      84      86      96      98     112
    SWAP_INTS       38      59     138
    T       47      72
    a       40+     42+     45+    130     134+    146     149+
    ap             144     146
    b       40+     42+     45+    130     134+    146     149+
    base            57      71      74      76      78+     79      83      84      85      86      90
            93      98      99     100     102     110     113     118
    basep           51      71
    bp             144     146
    comp            45      52
    h       24
    i       55      79      81      82      85      86      89      90      94      96      99     102
           104     110+
    j       55      80      81      82      83      84      92      93      94      96      98      99
           101     106     110+    111+    112+    113
    limit           55      74      76      78      80      99     105     106     110     119
    nbytes         130     135
    nelems          51      74
    nints          144     150
    qsort           28      51
    size            42      51      63      65      72      74      78+     79      80      89      92
          110+    111+    112
    size_t          28+     30      39      51+     56      60      61     130     144
    sp              54      73     100     101     104     105     108     116     117     118     119
    stack           54      73     116
    stddef          24
    swap_chars              30      42      64     130
    swap_func       40      61      64      67
    swap_ints       39      67     144
    thresh          56      72      76
    tmp            132     134+    147     149+
    width           40      60      63      66
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  longrand() -- generate 2**31-2 random numbers
**
**  public domain by Ray Gardner
**
**  based on "Random Number Generators: Good Ones Are Hard to Find",
**  S.K. Park and K.W. Miller, Communications of the ACM 31:10 (Oct 1988),
**  and "Two Fast Implementations of the 'Minimal Standard' Random
**  Number Generator", David G. Carta, Comm. ACM 33, 1 (Jan 1990), p. 87-88
**
**  linear congruential generator f(z) = 16807 z mod (2 ** 31 - 1)
**
**  uses L. Schrage's method to avoid overflow problems
*/

#define a 16807         /* multiplier */
#define m 2147483647L   /* 2**31 - 1 */
#define q 127773L       /* m div a */
#define r 2836          /* m mod a */

long nextlongrand(long seed)
{
      unsigned long lo, hi;

      lo = a * (long)(seed & 0xFFFF);
      hi = a * (long)((unsigned long)seed >> 16);
      lo += (hi & 0x7FFF) << 16;
      if (lo > m)
      {
            lo &= m;
            ++lo;
      }
      lo += hi >> 15;
      if (lo > m)
      {
            lo &= m;
            ++lo;
      }
      return (long)lo;
}

static long randomnum = 1;

long longrand(void)                     /* return next random long */
{
      randomnum = nextlongrand(randomnum);
      return randomnum;
}

void slongrand(unsigned long seed)      /* to seed it */
{
      randomnum = seed ? (seed & m) : 1;  /* nonzero seed */
}


#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      long reps, k, num;
      unsigned long seed;

      reps = 10000;
      seed = 1;

      /*
      ** correctness test: after 10000 reps starting with seed 1,
      ** result should be 1043618065
      */

      if (argc > 1)
            reps = atol(argv[1]);
      if (argc > 2)
            seed = atol(argv[2]);

      printf("seed %ld for %ld reps...\n", seed, reps);
      slongrand(seed);
      for (k = 0; k < reps; ++k)
            num = longrand();
      printf("%ld\n", num);

      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2092 characters
    90 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     2 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    TEST            58
    a       18      27      28
    argc            63      76      78
    argv            63      77      79
    atol            77      79
    h       60      61
    hi              25      28      29      35
    k       65      83+
    lo              25      27      29      30      32      33      35      36      38      39      41
    longrand        46      84
    m       19      30      32      36      38      54
    main            63
    nextlongrand            23      48
    num             65      84      85
    printf          81      85
    q       20
    r       21
    randomnum       44      48+     49      54
    reps            65      68      77      81      83
    seed            23      27      28      52      54+     66      69      79      81      82
    slongrand       52      82
    stdio           60
    stdlib          61
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  ssort()  --  Fast, small, qsort()-compatible Shell sort
  5 | **
  6 | **  by Ray Gardner,  public domain   5/90
  7 | */
  8 |
  9 | #include <stddef.h>
 10 | #include "snipsort.h"
 11 |
 12 | void ssort (void  *base,
 13 |             size_t nel,
 14 |             size_t width,
 15 |             int (*comp)(const void *, const void *))
 16 | {
 17 |       size_t wnel, gap, wgap, i, j, k;
 18 |       char *a, *b, tmp;
 19 |
 20 |       wnel = width * nel;
 21 |       for (gap = 0; ++gap < nel;)
 22 |             gap *= 3;
 23 |       while ( gap /= 3 )
 24 |       {
 25 |             wgap = width * gap;
 26 |             for (i = wgap; i < wnel; i += width)
 27 |             {
 28 |                   for (j = i - wgap; ;j -= wgap)
 29 |                   {
 30 |                         a = j + (char *)base;
 31 |                         b = a + wgap;
 32 |                         if ( (*comp)(a, b) <= 0 )
 33 |                               break;
 34 |                         k = width;
 35 |                         do
 36 |                         {
 37 |                               tmp = *a;
 38 |                               *a++ = *b;
 39 |                               *b++ = tmp;
 40 |                         } while ( --k );
 41 |                         if (j < wgap)
 42 |                               break;
 43 |                   }
 44 |             }
 45 |       }
 46 | }
```

TEXT STATISTICS

```
  1258 characters
    46 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    a       18      30      31      32      37      38
    b       18      31      32      38      39
    base            12      30
    comp            15      32
    gap             17      21+     22      23      25
    h        9
    i       17      26+     28
    j       17      28+     30      41
    k       17      34      40
    nel             13      20      21
    size_t          13      14      17
    ssort           12
    stddef           9
    tmp             18      37      39
    wgap            17      25      26      28+     31      41
    width           14      20      25      26      34
    wnel            17      20      26
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Originally published as part of the MicroFirm Function Library
  5 | **
  6 | **  Copyright 1986, S.E. Margison
  7 | **  Copyright 1989, Robert B.Stout
  8 | **
  9 | **  The user is granted a free limited license to use this source file
 10 | **  to create royalty-free programs, subject to the terms of the
 11 | **  license restrictions specified in the LICENSE.MFL file.
 12 | **
 13 | **  remove all whitespace from a string
 14 | */
 15 |
 16 | #include <stdio.h>
 17 | #include <ctype.h>
 18 | #include "snip_str.h"
 19 |
 20 | #if defined(__cplusplus) && __cplusplus
 21 |  extern "C" {
 22 | #endif
 23 |
 24 | char *rmallws(char *str)
 25 | {
 26 |       char *obuf, *nbuf;
 27 |
 28 |       if (str)
 29 |       {
 30 |             for (obuf = str, nbuf = str; *obuf; ++obuf)
 31 |             {
 32 |                   if (!isspace(*obuf))
 33 |                         *nbuf++ = *obuf;
 34 |             }
 35 |             *nbuf = NUL;
 36 |       }
 37 |       return str;
 38 | }
 39 |
 40 | #if defined(__cplusplus) && __cplusplus
 41 |  }
 42 | #endif
```

TEXT STATISTICS

```
   939 characters
    42 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    NUL             35
    __cplusplus             20+     40+
    ctype           17
    defined         20      40
    h       16      17
    isspace         32
    nbuf            26      30      33      35
    obuf            26      30+     32      33
    rmallws         24
    stdio           16
    str             24      28      30+     37
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Originally published as part of the MicroFirm Function Library
  5 | **
  6 | **  Copyright 1986, S.E. Margison
  7 | **  Copyright 1989, Robert B.Stout
  8 | **
  9 | **  The user is granted a free limited license to use this source file
 10 | **  to create royalty-free programs, subject to the terms of the
 11 | **  license restrictions specified in the LICENSE.MFL file.
 12 | **
 13 | **  remove leading whitespace from a string
 14 | */
 15 |
 16 | #include <ctype.h>
 17 | #include <string.h>
 18 | #include "snip_str.h"                    /* Contains strMove() macro   */
 19 |
 20 | #if defined(__cplusplus) && __cplusplus
 21 |  extern "C" {
 22 | #endif
 23 |
 24 | char *rmlead(char *str)
 25 | {
 26 |       char *obuf;
 27 |
 28 |       if (str)
 29 |       {
 30 |             for (obuf = str; *obuf && isspace(*obuf); ++obuf)
 31 |                   ;
 32 |             if (str != obuf)
 33 |                   strMove(str, obuf);
 34 |       }
 35 |       return str;
 36 | }
 37 |
 38 | #if defined(__cplusplus) && __cplusplus
 39 |  }
 40 | #endif
```

TEXT STATISTICS

```
   947 characters
    40 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    __cplusplus             20+     38+
    ctype           16
    defined         20      38
    h       16      17
    isspace         30
    obuf            26      30+     32      33
    rmlead          24
    str             24      28      30      32      33      35
    strMove         33
    string          17
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Originally published as part of the MicroFirm Function Library
  5 | **
  6 | **  Copyright 1986, S.E. Margison
  7 | **  Copyright 1989, Robert B.Stout
  8 | **
  9 | **  The user is granted a free limited license to use this source file
 10 | **  to create royalty-free programs, subject to the terms of the
 11 | **  license restrictions specified in the LICENSE.MFL file.
 12 | **
 13 | **  remove trailing whitespace from a string
 14 | */
 15 |
 16 | #include <string.h>
 17 | #include <ctype.h>
 18 | #include "snip_str.h"
 19 |
 20 | #if defined(__cplusplus) && __cplusplus
 21 |  extern "C" {
 22 | #endif
 23 |
 24 | char *rmtrail(char *str)
 25 | {
 26 |       int i;
 27 |
 28 |       if (str && 0 != (i = strlen(str)))
 29 |       {
 30 |             while (--i >= 0)
 31 |             {
 32 |                   if (!isspace(str[i]))
 33 |                         break;
 34 |             }
 35 |             str[++i] = NUL;
 36 |       }
 37 |       return str;
 38 | }
 39 |
 40 | #if defined(__cplusplus) && __cplusplus
 41 |  }
 42 | #endif
```

TEXT STATISTICS

```
   926 characters
    42 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     2 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    NUL             35
    __cplusplus             20+     40+
    ctype           17
    defined         20      40
    h       16      17
    i       26      28      30      32      35
    isspace         32
    rmtrail         24
    str             24      28+     32      35      37
    string          16
    strlen          28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Remove all files and (optionally) subdirectories
**
**  public domain demo by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <dos.h>
#include <ctype.h>
#include "sniptype.h"
#include "dirport.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

#define MAX_PATH 80

#if (defined(_MSC_VER) && (_MSC_VER >= 700)) || (defined(__SC__))
 /* Make FP_xxx macros lvalues as in older versions */
 #undef FP_SEG
 #undef FP_OFF
 #define FP_SEG(fp)    ((unsigned)((unsigned long)(fp) >> 16))
 #define FP_OFF(fp)    ((unsigned)(fp && 0xffff))
#endif

/* Select one of the following - remove() is ANSI       */

#define rmfunc remove
/* #define rmfunc unlink */

#define show(s) fputs((s), stderr)

Boolean_T recurse = False_, gobble = False_, ignore = False_;

char *mask = "*.*";

/*
**  Clean all files from a directory
*/

void clean_dir(char *path)
{
      char rmpath[MAX_PATH], *rmfile;
      DOSFileData fbuf;
      unsigned attrib = (ignore) ? _A_ANY : 0;

      strcpy(rmpath, path);
      if ('\\' != LAST_CHAR(rmpath))
            strcat(rmpath, "\\");
      rmfile = &rmpath[strlen(rmpath)];
      strcpy(rmfile, mask);
      if (0 == FIND_FIRST(rmpath, attrib, &fbuf)) do
      {
            strcpy(rmfile, ff_name(&fbuf));
            if (ignore)
            {
                  union REGS regs;
                  struct SREGS sregs;

                  regs.x.ax = 0x4300;
                  regs.x.dx = FP_OFF((char FAR *)rmpath);
                  segread(&sregs);
                  sregs.ds  = FP_SEG((char FAR *)rmpath);
                  intdosx(&regs, &regs, &sregs);
                  if (!regs.x.cflag)
                  {
                        regs.x.ax  = 0x4301;
                        regs.x.cx &= ~(_A_RDONLY | _A_HIDDEN | _A_SYSTEM);
                        intdosx(&regs, &regs, &sregs);
                        if (regs.x.cflag)
                              printf("unable to delete %s\n", rmpath);
                  }
            }
            rmfunc(rmpath);
            printf("deleting %s\n", rmpath);
      } while (0 == FIND_NEXT(&fbuf));
}

/*
**  Process directories
*/

void do_dir(char *path)
{
      char search[MAX_PATH], new[MAX_PATH];
      DOSFileData ff;

      strcpy(search, path);
      if ('\\' != LAST_CHAR(search))
            strcat(search, "\\");
      strcat(search, "*.*");
      if (Success_ == FIND_FIRST(search, _A_ANY, &ff)) do
      {
```

```
101 |             if (ff_attr(&ff) & _A_SUBDIR && '.' != *ff_name(&ff))
102 |             {
103 |                   strcpy(new, path);
104 |                   if ('\\' != LAST_CHAR(new))
105 |                         strcat(new, "\\");
106 |                   strcat(new, ff_name(&ff));
107 |                   do_dir(new);
108 |             }
109 |       } while (Success_ == FIND_NEXT(&ff));
110 |       clean_dir(path);
111 |       if (gobble)
112 |             rmdir(path);
113 | }
114 |
115 | /*
116 | **  Tell 'em they messed up
117 | */
118 |
119 | void usage(Boolean_T errstat)
120 | {
121 |       if (errstat)
122 |             fputc('\a', stderr);
123 |       show("Usage: RM_ALL directory [...directory] [-eFNAME.EXT] [-rgi?]\n");
124 |       show("switches: -eFNAME.EXT  Remove only files matching mask "
125 |             "(default is \"-e*.*\")\n");
126 |       show("          -r           Recurse subdirectories\n");
127 |       show("          -g           Gobble (delete) empty subdirectories\n");
128 |       show("          -i           Ignore special file attributes "
129 |             "(CAUTION!)\n");
130 |       show("          -?           Display help (this message)\n");
131 |       exit(errstat);
132 | }
133 |
134 | /*
135 | **  RM_ALL - Deletes all files and (optionally) subdirectories
136 | */
137 |
138 | int main(int argc, char *argv[])
139 | {
140 |       int i, j;
141 |       Boolean_T found_dir = False_;
142 |       void (*clean_func)(char *) = clean_dir;
143 |
144 |       for (i = 1; i < argc; ++i)          /* Check for switches         */
145 |       {
146 |             if (NULL == strchr("-/", *argv[i]))
147 |                   continue;               /* Assume it's a filename     */
148 |             for (j = 1; argv[i][j] ; ++j) /* Traverse nested switches   */
149 |             {
150 |                   switch (toupper(argv[i][j]))
151 |                   {
152 |                   case 'R':
153 |                         clean_func = do_dir;
154 |                         break;
155 |
156 |                   case 'G':
157 |                         gobble = True_;
158 |                         break;
159 |
160 |                   case 'I':
161 |                         ignore = True_;
162 |                         break;
163 |
164 |                   case '?':
165 |                         puts("***help***");
166 |                         usage(False_);
167 |                         break;
168 |
169 |                   case 'E':
170 |                         if (0 == strlen(&argv[i][++j]))
171 |                         {
172 |                               puts("***no file***");
173 |                               usage(Error_);              /* Oops     */
174 |                         }
175 |                         mask = strupr(&argv[i][j]);
176 |                         j += strlen(&argv[i][j]) - 1; /* End of switch  */
177 |                         break;
178 |
179 |                   default:
180 |                         puts("***default***");
181 |                         usage(Error_);
182 |                   }
183 |             }
184 |       }
185 |       for (i = 1; i < argc; ++i)          /* Scan filenames             */
186 |       {
187 |             if (strchr("/-", *argv[i]))
188 |                   continue;
189 |             found_dir = True_;
190 |             clean_func(argv[i]);
191 |       }
192 |       if (!found_dir)
193 |       {
194 |             puts("***not found***");
195 |             usage(True_);
196 |       }
197 |       return 0;
198 | }
```

TEXT STATISTICS

5710 characters
198 lines

LEXICAL STATISTICS

15 comments [std-C]
0 comments [C++]
22 preprocessor instructions
10 constants [character]
24 constants [string]
15 constants [numeric]

SYMBOL TABLE

```
    Boolean_T       40     119     141
    DOSFileData             51      93
    Error_         173     181
    FAR             68      70
    FIND_FIRST              59      99
    FIND_NEXT       83     109
    FP_OFF          28      30      68
    FP_SEG          27      29      70
    False_          40+    141     166
    LAST_CHAR       55      96     104
    MAX_PATH        23      50      92+
    MSDOS           17
    NULL           146
    REGS            64
    SREGS           65
    Success_        99     109
    True_          157     161     189     195
    _A_ANY          52      99
    _A_HIDDEN       75
    _A_RDONLY       75
    _A_SUBDIR      101
    _A_SYSTEM       75
    _MSC_VER        25+
    __MSDOS__       17
    __SC__          25
    argc           138     144     185
    argv           138     146     148     150     170     175     176     187     190
    attrib          52      59
    ax              67      74
    cflag           72      77
    clean_dir       48     110     142
    clean_func              142     153     190
    ctype           14
    cx              75
    defined         17+     25+
    do_dir          90     107     153
    dos             13
    ds              70
    dx              68
    errstat        119     121     131
    exit           131
    fbuf            51      59      61      83
    ff              93      99     101+    106     109
    ff_attr        101
    ff_name         61     101     106
    found_dir      141     189     192
    fp              29+     30+
    fputc          122
    fputs           38
    gobble          40     111     157
    h        9      10      11      12      13      14      20
    i      140     144+    146     148     150     170     175     176     185+    187     190
    ignore          40      52      62     161
    intdosx         71      76
    io              12
    j      140     148+    150     170     175     176+
    main           138
    mask            42      58     175
    path            48      54      90      95     103     110     112
    printf          78      82
    puts           165     172     180     194
    recurse         40
    regs            64      67      68      71+     72      74      75      76+     77
    remove          35
    rmdir          112
    rmfile          50      57      58      61
    rmfunc          35      81
    rmpath          50      54      55      56      57+     59      68      70      78      81      82
    s       38+
    search          92      95      96      97      98      99
    segread         69
    show            38     123     124     126     127     128     130
    sregs           65      69      70      71      76
    stderr          38     122
    stdio            9
    stdlib          10
    strcat          56      97      98     105     106
    strchr         146     187
    strcpy          54      58      61      95     103
    string          11
    strlen          57     170     176
    strupr         175
    toupper        150
    unistd          20
```

```
usage          119     166     173     181     195
x       67      68      72      74      75      77
```

```c
/* +++Date last modified: 05-Jul-1997 */

/*
**  RND_DIV.C - Rounded integer division
**
**  Public domain - suggested by Dave Hansen in comp.lang.c
*/

#include <stdlib.h>

int round_div(int n, int d)
{
      div_t res = div(n,d);
      div_t rnd = div(res.rem, (abs(d)+1)/2 );

      return res.quot + rnd.quot;
}

long round_ldiv(long n, long d)
{
      ldiv_t res = ldiv(n,d);
      ldiv_t rnd = ldiv(res.rem, (abs(d)+1)/2 );

      return res.quot + rnd.quot;
}

#ifdef TEST

#include <stdio.h>
#include <limits.h>

main(int argc, char *argv[])
{
      long n, d, q;

      if (argc != 3)
      {
            puts("Usage: RND_DIV n d\n");
            puts("Returns n/d rounded to nearest integer");
            return -1;
      }

      n = atol(argv[1]);
      d = atol(argv[2]);

      if (n > INT_MAX || d > INT_MAX)
      {
            q = round_ldiv(n, d);
            printf("round_ldiv(%ld, %ld) = %ld\n", n, d, q);
      }
      else
      {
            q = (long)round_div((int)n, (int)d);
            printf("round_div(%ld, %ld) = %ld\n", n, d, q);
      }

      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1170 characters
    60 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     4 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    INT_MAX          46+
    TEST             27
    abs              14      22
    argc             32      36
    argv             32      43      44
    atol             43      44
    d        11      13      14      19      21      22      34      44      46      48      49      53
             54
    div              13      14
    div_t            13      14
    h         9      29      30
    ldiv             21      22
    ldiv_t           21      22
    limits           30
    main             32
    n        11      13      19      21      34      43      46      48      49      53      54
    printf           49      54
    puts             38      39
    q        34      48      49      53      54
    quot             16+     24+
    rem              14      22
    res              13      14      16      21      22      24
    rnd              14      16      22      24
    round_div        11      53
    round_ldiv               19      48
    stdio            29
    stdlib            9
```

```
/* +++Date last modified: 05-Jul-1997 */

/*_|_|   ROLLDICE.C
_|_|_|   A function to roll a definable number of dice (1 - 100),
_|_|_|   with a definable number of sides (2 to 255).
_|_|_|   No warrantees or guarantees are given or implied.
_|_|_|   Released to   PUBLIC DOMAIN   by Kurt Kuzba.  (8/25/96)
*/

#include "rolldice.h"                    /* Also includes stdlib.h     */

static int rnd(int min, int max)
{
      long t = (long)rand() * (long)(max - min + 1);
      return min + (int)(t / RAND_MAX);
}

char *roll_dice(int d, int s)
{
      char *result = malloc(d * s + 1);
      int shuffle_loop, die, side, temp, random;

      if (result == NULL)
            return NULL;
      *(result + d * s) = '\0';

      /* initialize your dice */

      for (die = 0; die < d; die++)
      {
            for (side = 0; side < s; side++)
                  *(result + die * s + side) = (char)(side + 1);
      }
      for (shuffle_loop = 0; shuffle_loop < 16; shuffle_loop ++)
      {
            /* 'roll' your dice     */

            for (die = 0; die < d; die++)
            {
                  for (side = 0; side < s; side++)
                  {
                        random = rnd(1, s) - 1;
                        temp = *(result + die * s + side);
                        *(result + die * s + side) =
                              *(result + die * s + random);
                        *(result + die * s + random) = (char)temp;
                  }
            }
      }
      for (die = 0; die < d; die++)    /* Make an 'upside' string */
            *(result + die) = *(result + die * s);
      *(result + d) = '\0';           /* Trim str for # of dice  */
      return result;
}

#ifdef TEST

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <conio.h>

int main(int c, char *v[])
{
      time_t t;
      int dice, sides;
      char *d;

      srand((unsigned)time(&t));
      dice = atoi(v[1]);
      d = strchr(v[1], 'd');
      sides = atoi(++d);
      if (d == NULL || dice < 1 || dice > 100 || sides < 2 || sides > 255)
            return !!puts("USAGE EX: ROLLDICE 2d6   (rolls 2 6-sided dice)");
      d = roll_dice(dice, sides);
      if (d == NULL)
            return !puts("Malloc Failure.");
      dice = sides = 0;
      while (d[dice])
      {
            sides += d[dice];
            printf("%d ", d[dice++]);
      }
      printf("\nYour total = %d\n", sides);
      free(d);   /* Make sure you free your dice after every usage!! */
      return 0;  /* Otherwise, you will leak away all your memory!!! */
}

#endif /* TEST */

/*_|_|   end   ROLLDICE.C   */
```

TEXT STATISTICS

```
  2701 characters
    91 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     3 constants [character]
     5 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    NULL            23      24      73      76
    RAND_MAX        15
    TEST            56
    atoi            70      72
    c       63
    conio           61
    d       18      20      25      29      38      50      52      67      71      72      73      75
            76      79      81      82      85
    dice            66      70      73+     75      78      79      81      82
    die             21      29+     32      38+     43      44      45      46      50+     51+
    free            85
    h       58      59      60      61
    main            63
    malloc          20
    max             12      14
    min             12      14      15
    printf          82      84
    puts            74      77
    rand            14
    random          21      42      45      46
    result          20      23      25      32      43      44      45      46      51+     52      53
    rnd             12      42
    roll_dice       18      75
    s       18      20      25      31      32      40      42      43      44      45      46      51
    shuffle_loop            21      34+
    side            21      31+     32+     40+     43      44
    sides           66      72      73+     75      78      81      84
    srand           69
    stdio           58
    strchr          71
    string          60
    t       14      15      65      69
    temp            21      43      46
    time            59      69
    time_t          65
    v       63      70      71
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  ROLLDICE.H - SNIPPETS header file for ROLLDICE.C
   5 | */
   6 |
   7 | #ifndef ROLLDICE__H_
   8 | #define ROLLDICE__H_
   9 |
  10 | #include <stdlib.h>
  11 |
  12 | char *roll_dice(int d, int s);
  13 |
  14 | #endif /* ROLLDICE__H_ */
```

TEXT STATISTICS

```
   238 characters
    14 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    ROLLDICE__H_             7       8
    d       12
    h       10
    roll_dice       12
    s       12
    stdlib          10
```

```c
/* +++Date last modified: 05-Jul-1997 */

/*
**  ROMAN2L.C Covert roman numerals to long integers.
**
**  public domain by Bob Stout
*/

#include <ctype.h>
#include "sniptype.h"

struct numeral {
      long val;
      int  ch;
};

static struct numeral numerals[] = {
      {    1L, 'I' },
      {    5L, 'V' },
      {   10L, 'X' },
      {   50L, 'L' },
      {  100L, 'C' },
      {  500L, 'D' },
      { 1000L, 'M' }
};

/*
**  roman2long() - Converts a roman numeral string into a long integer
**
**  Arguments: 1 - Roman numeral string
**
**  Returns: Long value if valid, else -1L
*/

long roman2long(const char *str)
{
      int i, j, k;
      long retval = 0L;

      if (!str || NUL == *str)
            return -1L;
      for (i = 0, k = -1; str[i]; ++i)
      {
            for (j = 0; j < 7; ++j)
            {
                  if (numerals[j].ch == toupper(str[i]))
                        break;
            }
            if (7 == j)
                  return -1L;
            if (k >= 0 && k < j)
            {
                  retval -= numerals[k].val * 2;
                  retval += numerals[j].val;
            }
            else  retval += numerals[j].val;
            k = j;
      }
      return retval;
}

#ifdef TEST

#include <stdio.h>

int main(int argc, char *argv[])
{
      while (--argc)
      {
            ++argv;
            printf("roman2long(%s) returned %ld\n", *argv, roman2long(*argv));
      }
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1530 characters
    75 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     7 constants [character]
     2 constants [string]
    17 constants [numeric]
```

SYMBOL TABLE

NUL		40						
TEST		62						
argc		66	68					
argv		66	70	71+				
ch		14	46					
ctype		9						
h	9	64						
i	37	42+	46					
j	37	44+	46	49	51	54	56	57
k	37	42	51+	53	57			
main		66						
numeral		12	17					
numerals		17	46	53	54	56		
printf		71						
retval		38	53	54	56	59		
roman2long			35	71				
stdio		64						
str		35	40+	42	46			
toupper		46						
val		13	53	54	56			

```
/* +++Date last modified: 05-Jul-1997 */

/*
** rounding macros by Dave Knapp, Thad Smith, Jon Strayer, & Bob Stout
*/

#ifndef ROUND__H
#define ROUND__H

#include <math.h>

#if defined(__cplusplus) && __cplusplus

/*
** Safe C++ inline versions
*/

/* round to integer */

inline int iround(double x)
{
      return (int)floor(x + 0.5);
}

/* round number n to d decimal points */

inline double fround(double n, unsigned d)
{
      return floor(n * pow(10., d) + .5) / pow(10., d);
}

#else

/*
** NOTE: These C macro versions are unsafe since arguments are referenced
**       more than once.
**
**       Avoid using these with expression arguments to be safe.
*/

/*
** round to integer
*/

#define iround(x) floor((x) + 0.5)

/*
** round number n to d decimal points
*/

#define fround(n,d) (floor((n)*pow(10.,(d))+.5)/pow(10.,(d)))

#endif

#endif /* ROUND__H */
```

TEXT STATISTICS

```
   924 characters
    55 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     0 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    ROUND__H          7       8
    __cplusplus               12+
    d        27       29+     51+
    defined           12
    floor             22      29      45      51
    fround            27      51
    h        10
    iround            20      45
    math              10
    n        27       29      51+
    pow               29+     51+
    x        20       22      45+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  RULER.C(PP) - A utility to create text ruler lines
**
**  R.F. Pels. Dec. 1993. Placed in public domain.
*/

#include <string.h>
#include <stdio.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char * rule_line(char * s,
                 unsigned short len,
                 short units,
                 char * digits,
                 char filler)
{
      /*  If possible, initialize directly with correct value!    */
      short           whichdigit = 0;
      short           digitlen = strlen(digits);
      unsigned short  i;

      memset(s, filler, len);            /*  Fill string with all filler   */
      s[len]  = NUL;                     /*  Tack on an ASCIIZ             */

      for (i = 0; i < len; i += units)
      {
            s[i] = digits[whichdigit];    /*  Put in digit                  */
            whichdigit++;                 /*  Add 1 and reset to 0 if...    */
            whichdigit %= digitlen;       /* ...bigger than length of digits*/
      }
      return s;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

char text[41] = "abcdefghijabcdefhijkabcdefghijabcdefghij", ruler[41];

main(void)
{
      puts(text);
      printf("%s\n\n", rule_line(ruler, 40,   1, "123", ' '));
      puts(text);
      printf("%s\n\n", rule_line(ruler, 40,   1, "12345", ' '));
      puts(text);
      printf("%s\n\n", rule_line(ruler, 40,   1, "1234567890", ' '));
      puts(text);
      printf("%s\n\n", rule_line(ruler, 40,   5, "0123456789", ' '));
      puts(text);
      printf("%s\n\n", rule_line(ruler, 40,  10, "1234567890", ' '));
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1769 characters
    63 lines

LEXICAL STATISTICS

     9 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     5 constants [character]
    13 constants [string]
    15 constants [numeric]

SYMBOL TABLE

    NUL             29
    TEST            44
    __cplusplus             13+     40+
    defined         13      40
    digitlen        25      35
    digits          20      25      33
    filler          21      28
    h        9      10
    i       26      31+     33
    len             18      28      29      31
    main            48
    memset          28
    printf          51      53      55      57      59
    puts            50      52      54      56      58
    rule_line       17      51      53      55      57      59
    ruler           46      51      53      55      57      59
    s       17      28      29      33      37
    stdio           10
    string           9
    strlen          25
    text            46      50      52      54      56      58
    units           19      31
    whichdigit              24      33      34      35
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    SAFEMINX.c  Safe Multiple Includes.

                A utility to prevent problems with multiple #include's of
                the same file. Which is a severe problem for some older
                compilers like Turbo C version 1.5 ...

                95-10-26  v1.00  Initial version

 _____          This version is Public Domain.
/_|__|          A.Reitsma, Delft, Nederland.
/  |  \ ------------------------------------------------------------ */

#include <stdlib.h>     /* for EXIT_SUCCESS              */
#include <stdio.h>      /* for fprintf()/fopen()/fclose() */
#include <string.h>     /* for strlen etc ...            */
#include <ctype.h>      /* for isalpha()                 */
#include "dirport.h"

#ifndef FILENAME_MAX
  #define FILENAME_MAX  80
#endif
#ifndef FNAME_MAX
  #define FNAME_MAX     8+3+1
#endif

#define PROGRAM   "SAFEMINX"
#define VERSION   "v1.00"
#define AUTHOR    "A.Reitsma, Delft, The Netherlands."
#define CREATED   "95-10-26"
#define COPYRIGHT "Public Domain."

char * Syntax[] =
{
    PROGRAM "  " VERSION "  " CREATED "  " AUTHOR "  " COPYRIGHT "\n\n",
    "Usage: " PROGRAM " <destination directory name>\n\n",
    "The problem:\n",
    "Some old compilers have severe problems with multiple #include's\n",
    "of the same file. This program eliminates the problem by creating\n",
    "'include-file' wrappers in the destination directory for the\n",
    "include files in the current directory.\n",
    "The wrappers could also be edited to include items which _should_\n",
    "be in the include file according to the standard but aren't.\n",
    "E.g.: FILENAME_MAX in stdio.h or CLOCKS_PER_SEC in time.h\n\n",
    "Instructions:\n- create a new include file directory,\n",
    "- change directory to the old include file directory,\n",
    "- run this program with the new directory as parameter,\n",
    "- change your compiler's settings to use the new directory.\n",
    "(Or something to the same effect.)\n\n",
    "Files are NEVER deleted or modified!\n\n",
    "Problem permanently solved ...\n",
    NULL
};

enum Errors
{
    ERR_ARGS = 1,
    ERR_MEM,
    ERR_FOPEN,
    ERR_NODIR,
    ERR_INVDRIVE,
    ERR_DIRCURRENT,
    ERR_NOFILES,
};

main( int argc, char * argv[] )
{
    char DirCurrent[ FILENAME_MAX ];
    char DestName[ FILENAME_MAX ];
    int DestDirLen;
    DOSFileData Data;

    /* validate the arguments
    */
    if( argc < 2 )
    {
        int ix = 0;
        do
        {
            fprintf( stdout, Syntax[ ix ]);
        }while( NULL != Syntax[ ++ix ]);
        return ERR_ARGS;
    }

    /* obtain the name of the current directory
    */
    getcwd( DirCurrent, FILENAME_MAX );
    strlwr( DirCurrent );

    /* validate the destination drive and directory
    */
    strcpy( DestName, argv[ 1 ]);
    strlwr( DestName );
    if( ':' == DestName[ 1 ] )
    {
        DestName[ 2 ] = '.';
        DestName[ 3 ] = '\0';
        if( 0 != chdir( DestName ))
```

```
101 |         {
102 |             fprintf( stderr, "\aInvalid destination drive\n" );
103 |             return ERR_INVDRIVE;
104 |         }
105 |         DestName[ 2 ] = argv[ 1 ][ 2 ];
106 |         DestName[ 3 ] = argv[ 1 ][ 3 ];
107 |     }
108 |     if( 0 != chdir( argv[ 1 ]) )
109 |     {
110 |         fprintf( stderr, "\aInvalid destination directory\n" );
111 |         return ERR_NODIR;
112 |     }
113 |
114 |     chdir( DirCurrent ); /* just in case it was changed ... */
115 |
116 |     if( 0 == strcmp( DirCurrent, DestName ))
117 |     {
118 |         fprintf( stderr, "\aDestination is current directory.\n" );
119 |         return ERR_DIRCURRENT;
120 |     }
121 |
122 |     /* anything to do?
123 |     */
124 |     if( 0 != FIND_FIRST( "*.h", _A_NORMAL, &Data ))
125 |     {
126 |         fprintf( stderr, "\aNo .h files in the current directory\n" );
127 |         return ERR_NOFILES;
128 |     }
129 |
130 |     /* Yes, something to do: prepare for action
131 |     */
132 |     fprintf( stdout, "Directory: \"%s\" ---\n", DirCurrent );
133 |     DestDirLen = strlen( argv[ 1 ]);
134 |     memcpy( DestName, argv[ 1 ], DestDirLen+1 );
135 |     if( '\\' != argv[ 1 ][ DestDirLen - 1 ] )
136 |     {
137 |         DestName[ DestDirLen ] = '\\' ;
138 |         DestDirLen++;
139 |     }
140 |
141 |     do
142 |     {
143 |         char NameAlpha[ FNAME_MAX+1 ];
144 |         char * NamePtr = NameAlpha;
145 |         FILE * FileOut;
146 |
147 |         strcpy( DestName+DestDirLen, ff_name( &Data ));
148 |
149 |         FileOut = fopen( DestName, "r" );
150 |         if( NULL != FileOut )
151 |         {
152 |             fprintf( stdout,
153 |                      "Not overwriting \"%s\". It already exists!\n",
154 |                      ff_name( &Data ));
155 |             continue;
156 |         }
157 |         FileOut = fopen( DestName, "w" );
158 |         if( NULL == FileOut )
159 |         {
160 |             fprintf( stdout, "\aFailure opening \"%s\".\n",
161 |                              ff_name( &Data ));
162 |             continue;
163 |         }
164 |
165 |         fprintf( stdout, "Creating \"%s\"\n", DestName );
166 |
167 |         strcpy( NameAlpha, ff_name( &Data ));
168 |         while( '\0' != *NamePtr )
169 |         {
170 |             if( !isalpha( *NamePtr ))
171 |                  *NamePtr = '_';
172 |             NamePtr ++ ;
173 |         }
174 |         fprintf( FileOut,       /* fixed text: */
175 |                 "/*  %s\n"
176 |                 "    Prevention of double inclusions.\n"
177 |                 "*/\n\n"
178 |                 "#ifndef SAFE__%s\n"
179 |                 "#define SAFE__%s\n\n"
180 |                 "#include <%s\\%s>    /* 'original' */\n"
181 |                 "\n#endif\n",
182 |                                 /* parameters: */
183 |                 ff_name( &Data ),
184 |                 NameAlpha,
185 |                 NameAlpha,
186 |                 DirCurrent,
187 |                 ff_name( &Data )
188 |                );
189 |
190 |         fclose( FileOut );
191 |     }while( 0 == FIND_NEXT( &Data ));
192 |     FIND_END( &Data );
193 |
194 |     fprintf( stdout, "--- Done ---\n" );
195 |     return EXIT_SUCCESS;
196 | }
197 |
198 | /* === SAFEMINX.c end ============================================== */
```

TEXT STATISTICS

```
  6018 characters
   198 lines
```

LEXICAL STATISTICS

```
    15 comments [std-C]
     0 comments [C++]
    16 preprocessor instructions
     7 constants [character]
    47 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    AUTHOR          31      37
    COPYRIGHT       33      37
    CREATED         32      37
    DOSFileData             73
    Data            73     124     147     154     161     167     183     187     191     192
    DestDirLen              72     133     134     135     137     138     147
    DestName        71      94      95      96      98      99     100     105     106     116     134
           137     147     149     157     165
    DirCurrent              70      89      90     114     116     132     186
    ERR_ARGS        59      84
    ERR_DIRCURRENT          64     119
    ERR_FOPEN       61
    ERR_INVDRIVE            63     103
    ERR_MEM         60
    ERR_NODIR       62     111
    ERR_NOFILES             65     127
    EXIT_SUCCESS           195
    Errors          57
    FILE           145
    FILENAME_MAX            22      23      70      71      89
    FIND_END       192
    FIND_FIRST              124
    FIND_NEXT      191
    FNAME_MAX       25      26     143
    FileOut        145     149     150     157     158     174     190
    NULL            54      83     150     158
    NameAlpha      143     144     167     184     185
    NamePtr        144     168     170     171     172
    PROGRAM         29      37      38
    Syntax          35      82      83
    VERSION         30      37
    _A_NORMAL      124
    argc            68      77
    argv            68      94     105     106     108     133     134     135
    chdir          100     108     114
    ctype           19
    fclose         190
    ff_name        147     154     161     167     183     187
    fopen          149     157
    fprintf         82     102     110     118     126     132     152     160     165     174     194
    getcwd          89
    h       16      17      18      19
    isalpha        170
    ix              79      82      83
    main            68
    memcpy         134
    stderr         102     110     118     126
    stdio           17
    stdlib          16
    stdout          82     132     152     160     165     194
    strcmp         116
    strcpy          94     147     167
    string          18
    strlen         133
    strlwr          90      95
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** scalar date routines    --    public domain by Ray Gardner
** Numerically, these will work over the range 1/01/01 thru 14699/12/31.
** Practically, these only work from the beginning of the Gregorian
** calendar thru 14699/12/31.  The Gregorian calendar took effect in
** much of Europe in about 1582, some parts of Germany in about 1700, in
** England and the colonies in about 1752ff, and in Russia in 1918.
*/

#include "scaldate.h"

int isleap (unsigned yr)
{
   return yr % 400 == 0 || (yr % 4 == 0 && yr % 100 != 0);
}

static unsigned months_to_days (unsigned month)
{
   return (month * 3057 - 3007) / 100;
}

static long years_to_days (unsigned yr)
{
   return yr * 365L + yr / 4 - yr / 100 + yr / 400;
}

long ymd_to_scalar (unsigned yr, unsigned mo, unsigned day)
{
   long scalar;
   scalar = day + months_to_days(mo);
   if ( mo > 2 )                        /* adjust if past February */
      scalar -= isleap(yr) ? 1 : 2;
   yr--;
   scalar += years_to_days(yr);
   return scalar;
}

void scalar_to_ymd (long scalar, unsigned *yr, unsigned *mo, unsigned *day)
{
   unsigned n;               /* compute inverse of years_to_days() */

   for ( n = (unsigned)((scalar * 400L) / 146097L); years_to_days(n) < scalar;)
      n++;                           /* 146097 == years_to_days(400) */
   *yr = n;
   n = (unsigned)(scalar - years_to_days(n-1));
   if ( n > 59 ) {                      /* adjust if past February */
      n += 2;
      if ( isleap(*yr) )
         n -= n > 62 ? 1 : 2;
   }
   *mo = (n * 100 + 3007) / 3057;    /* inverse of months_to_days() */
   *day = n - months_to_days(*mo);
}
```

TEXT STATISTICS

```
  1718 characters
    55 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    day             29      32      40      54
    isleap          14      34      50
    mo              29      32      33      40      53      54
    month           19      21
    months_to_days          19      32      54
    n       42      44+     45      46      47+     48      49      51+     53      54
    scalar          31      32      34      36      37      40      44+     47
    scalar_to_ymd           40
    years_to_days           24      36      44      47
    ymd_to_scalar           29
    yr              14      16+     24      26+     29      34      35      36      40      46      50
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** scalar date routines    --    public domain by Ray Gardner
** Numerically, these will work over the range 1/01/01 thru 14699/12/31.
** Practically, these only work from the beginning of the Gregorian
** calendar thru 14699/12/31.  The Gregorian calendar took effect in
** much of Europe in about 1582, some parts of Germany in about 1700, in
** England and the colonies in about 1752ff, and in Russia in 1918.
*/

#ifndef SCALDATE__H
#define SCALDATE__H

#include "sniptype.h"

/*
**  Define ISO_CAL to be 1 for ISO (Mon-Sun) calendars
**
**  ISO defines the first week with 4 or more days in it to be week #1.
*/

#ifndef ISO_CAL
 #define ISO_CAL 0
#endif

#if (ISO_CAL != 0 && ISO_CAL != 1)
 #error ISO_CAL must be set to either 0 or 1
#endif

#if ISO_CAL
 enum DOW_T {DOW_IGNORE = -1,
        MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY};
#else
 enum DOW_T {DOW_IGNORE = -1,
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};
#endif

/*
**  Daylight savings time rules.
**
**  Rules include a month, date, and day. If the day is DOW_IGNORE, DST will
**  start on the month and date specified. If a day is specified (the
**  interpretation of the day parameter is subject to the value of ISO_CAL),
**  DST will start on the first such day following (or equal to) the specified
**  date, or stop on the first such day preceding (or equal to) the specified
**  date.
**
**  The defaults defined for the U.S. mean that DST will begin on the first
**  Sunday after (or on) April 1 and end on the last Sunday preceding (or on)
**  October 31.
*/

extern unsigned   DST_start_mo;
extern unsigned   DST_start_dt;
extern enum DOW_T DST_start_dy;

extern unsigned   DST_stop_mo;
extern unsigned   DST_stop_dt;
extern enum DOW_T DST_stop_dy;

int  isleap (unsigned yr);
long ymd_to_scalar (unsigned yr, unsigned mo, unsigned day);
void scalar_to_ymd (long scalar, unsigned *yr, unsigned *mo, unsigned *day);
int  daynum(int year, int month, int day);
int  weeknum(int year, int month, int day);

Boolean_T valiDate(unsigned yr, unsigned mo, unsigned day);

unsigned dow(unsigned yr, unsigned mo, unsigned day);
unsigned DOW(unsigned y, unsigned m, unsigned d);

long today(void);

extern char *MoonPhaseText[8];

unsigned moonphase(unsigned yr, unsigned mo, unsigned dy);

int getfdate (int handle, long *date);
int getdatef (char *fname, long *date);


#endif /* SCALDATE__H */
```

TEXT STATISTICS

```
  2541 characters
    83 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     1 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       68
    DOW             71
    DOW_IGNORE              32      35
    DOW_T           32      35      56      60
    DST_start_dt            55
    DST_start_dy            56
    DST_start_mo            54
    DST_stop_dt             59
    DST_stop_dy             60
    DST_stop_mo             58
    FRIDAY          33      36
    ISO_CAL         23      24      27+     28      31
    MONDAY          33      36
    MoonPhaseText           75
    SATURDAY        33      36
    SCALDATE__H             12      13
    SUNDAY          33      36
    THURSDAY        33      36
    TUESDAY         33      36
    WEDNESDAY       33      36
    be              28
    d       71
    date            79      80
    day             63      64      65      66      68      70
    daynum          65
    dow             70
    dy              77
    either          28
    fname           80
    getdatef        80
    getfdate        79
    handle          79
    isleap          62
    m       71
    mo              63      64      68      70      77
    month           65      66
    moonphase       77
    must            28
    scalar          64
    scalar_to_ymd           64
    set             28
    to              28
    today           73
    valiDate        68
    weeknum         66
    y       71
    year            65      66
    ymd_to_scalar           63
    yr              62      63      64      68      70      77
```

```
/* +++Date last modified: 05-Jul-1997 */

/* function scanfrac - scan an input string for a numeric value.
**
** Written in ANSI C and contributed to the public domain by
** Thad Smith III, Boulder, CO.     August 5, 1991
*/

/*****************************************************************
** scanfrac() scans an input string for a numeric value, which can
** be specified as:
**  1. an integer,              5
**  2. a floating point value,  5.1
**  3. a fraction, or           3/4
**  4. a mixed fraction.        5 3/4  or  5-3/4
**
** Conditions:
**  1. Preceding whitespace is allowed.
**  2. The input number may be signed.
**  3. The fractional part of a mixed fraction (but not pure fraction)
**     must be less than 1.
**  4. The numerator and denominator of a fraction or mixed fraction
**     must be less than 2^31.
**
** Parameters:
**  1. Input buffer containing value.
**  2. Pointer to double to receive return value.
**
** Return status:
**  0 = OK, value returned in f,
**  1 = bad input format,
**  2 = can't allocate memory
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "snipmath.h"

int scanfrac (const char buf[], double *f)
{
      char *tbuf = malloc (strlen(buf) +2);   /* input + terminator     */
      static char term[] = "\a";              /* terminator flag        */
      char t1,t2,t3;                          /* separator chars        */
      char sign;                              /* possible sign          */
      int nc;                                 /* # conversions          */
      long int b,c;                           /* 2nd & 3rd inputs       */

      if (!tbuf)                        /* couldn't allocate memory   */
            return 2;

      /* Copy the input to a temporary buffer and append a terminator
      ** character.  This terminator is used to determine whether the
      ** scanning of the input field by sscanf() was terminated by end
      ** of input or by an invalid character.  If terminated properly,
      ** the terminator character picked up in t1, t2, or t3.
      */

      strcat (strcpy(tbuf, buf), term);       /* input + term flag      */
      nc = sscanf (tbuf, " %lf%c %ld %c %ld %c",
                         f,&t1,&b,&t2,&c,&t3);
      free (tbuf);

      switch (nc)                   /* number of sscanf() conversions   */
      {
      case 2:         /* single floating value: a */
            if (t1 == *term) return 0;
            break;
      case 4:         /* pure fraction: a/b */
            if (t1 == '/' && t2 == *term && fmod (*f,1.0) == 0.0 && b > 0)
            {
                  *f /= b;
                  return 0;
            }
            break;
      case 6:         /* mixed fraction: a b/c  or  a-b/c */
            if ((t1 == ' ' || t1 == '-') && t2 == '/' && t3 == *term &&
                  fmod (*f,1.0) == 0.0 && b >= 0 && c > b)
            {
                  /* get first non-blank character so that
                  ** -0 b/c will be neg
                  */

#ifdef __ZTC__  /* fix for missing const in sscanf() declaration */
                  sscanf ((char*)buf, " %c", &sign);
#else
                  sscanf (buf, " %c", &sign);
#endif
                  if (sign == '-')
                        *f -= (double)b/c;
                  else  *f += (double)b/c;
                  return 0;
            }
      }
      return 1;
}

#ifdef TEST
```

```
 101 | /* This is a simple test driver.  It should be omitted before
 102 | ** placing scanfrac() into a library.
 103 | */
 104 |
 105 | main ()
 106 | {
 107 |       char buf[80];
 108 |       double f;
 109 |       int stat;
 110 |
 111 |       printf ("Enter 999. or generate EOF to stop\n");
 112 |       do
 113 |       {
 114 |             printf ("Enter value: ");
 115 |             if (! gets (buf))
 116 |             {
 117 |                   printf ("EOF detected. Aborting.\n");
 118 |                   return 1;
 119 |             }
 120 |             stat = scanfrac (buf, &f);
 121 |             printf ("\nStat = %d, value = %f\n", stat, f);
 122 |       } while ( f != 999.);
 123 |       return 0;
 124 | }
 125 |
 126 | #endif /* TEST */
```

TEXT STATISTICS

```
  4134 characters
   126 lines
```

LEXICAL STATISTICS

```
    20 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     5 constants [character]
     9 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    TEST            99
    __ZTC__         85
    b       48      62      71      73      79+     91      92
    buf             41      43      60      86      88     107     115     120
    c       48      62      79      91      92
    f       41      62      71      73      79      91      92     108     120     121     122
    fmod            71      79
    free            63
    gets           115
    h       35      36      37      38
    main           105
    malloc          43
    math            38
    nc              47      61      65
    printf         111     114     117     121
    scanfrac        41     120
    sign            46      86      88      90
    sscanf          61      86      88
    stat           109     120     121
    stdio           35
    stdlib          36
    strcat          60
    strcpy          60
    string          37
    strlen          43
    t1              45      62      68      71      78+
    t2              45      62      71      78
    t3              45      62      78
    tbuf            43      50      60      61      63
    term            44      60      68      71      78
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Macros for managing direct video writes by Jerry Houston
**
**  prototypes for SCROLL.C and VIDPORT.C functions added by Bob Stout
*/

#ifndef SCRNMACS__H
#define SCRNMACS__H

#include "sniptype.h"
#include "extkword.h"

#if !defined(COLORMODE)
 #define COLORMODE  ((*(char FAR *)0x0449L) != 7)
 #define EXT_KBD    (*(char FAR *)0x0496L & 16)
 #define VIDPAGE    (*((unsigned char far *)0x0462L))
 #define ROWSIZE    (*(int FAR *)0x044AL)
 #define SCANLINES  ((int)*(char FAR*)0x0461L)
 #define SCRBUFF    ((unsigned FAR *)((COLORMODE)?0xB8000000L:0xB0000000L))
 #define SCREENSEG  ((unsigned)((COLORMODE)?0xB800:0xB000))
 #define SCRNBYTES  (*(int FAR *)0x44CL)
 #define SCRNPIXELS (SCRNBYTES >> 1)
 #define SCREENCOLS (*(int FAR *)0x044AL)
 #define SCREENROWS ((*(char FAR *)0x0484L) ? 1 + (*(char FAR *)0x0484L): 25)
#endif

/*
     COLORMODE  = true/false, are we using color?
     EXT_KBD    = true/false, extended keyboard in use?
     VIDPAGE    = current video page in use
     SCANLINES  = number of scan lines in a character.
     SCRBUFF    = returns B800:0000 if using color, B000:0000 if mono.
     SCREENSEG  = when you just need the segment portion.
     SCRNBYTES  = number of bytes required to save screen.
     SCRNPIXELS = number of (2-byte) pixels required to save screen.
     SCREENCOLS = number of columns, often 80.
     SCREENROWS = number of rows, usually defaults to 25.
*/

/*
** video attributes
*/

#define BLINKING 0x87
#define REVERSE 0x70
#define REVBLINK 0xf0
#define NORMAL 0x07
#define HIGHLITE 0x0f
#define HIGHBLINK 0x8f
#define BLINKBIT 0x80   /* OR in to cause blink */
#define HILTBIT 0x08    /* OR in to cause highlight */

/*
** colors -- Use as is for foreground colors
**           For background, shift left by 4 and OR with
**           foreground and possible video attributes
*/

#define BLACK_ 0
#define BLUE_ 1
#define GREEN_ 2
#define CYAN_ 3
#define RED_ 4
#define MAGENTA_ 5
#define BROWN_ 6
#define WHITE_ 7
#define GRAY_ 8
#define LTBLUE_ 9
#define LTGREEN_ 10
#define LTCYAN_ 11
#define LTRED_ 12
#define LTMAGENTA_ 13
#define YELLOW_ 14
#define HIWHITE_ 15     /* hi-intensity white */

#define BG_(a) (((a) & 0x7f) << 4)

/*
** e.g. blue background with yellow characters
**
** video_attribute = BG_(BLUE_)+YELLOW_
*/

/*
**  File: SCROLL.C
*/

#define SCROLL_UP 0
#define SCROLL_DN 1

void scroll(int direction,
            int num_lines,
            int vattrib,
            int ulrow,
            int ulcomumn,
            int lrrow,
            int lrcolumn);

```

```
101 | /*
102 | **  File: VIDPORT.C
103 | */
104 |
105 | void GotoXY(int col, int row);
106 | void ClrScrn(int vattrib);
107 | void GetCurPos(int *col, int *row);
108 | int  GetCurAtr(void);
109 | void ClrEol(void);
110 | void ClrEop(void);
111 | void Repaint(int vattrib);
112 |
113 |
114 | /*
115 | **  File: SCNRSAVE.C
116 | */
117 |
118 | struct SCREEN {
119 |       unsigned short *vbuf;
120 |       int             curX,
121 |                       curY;
122 | };
123 |
124 | struct SCREEN *SaveScrn(void);
125 | void RestoreScrn(struct SCREEN *screen);
126 | void FreeScrnBuf(struct SCREEN *screen);
127 |
128 |
129 | /*
130 | **  File: FSCRNSAVE.C
131 | */
132 |
133 | Boolean_T fSaveScrn(const char *fname);
134 | Boolean_T fRestoreScrn(const char *fname);
135 |
136 | /*
137 | **  File: ATR2ANSI.C
138 | */
139 |
140 | char *make_ansi(int vatr);
141 |
142 | #endif /* SCRNMACS__H */
```

```
TEXT STATISTICS

  3367 characters
   142 lines

LEXICAL STATISTICS

    15 comments [std-C]
     0 comments [C++]
    45 preprocessor instructions
     0 constants [character]
     2 constants [string]
    46 constants [numeric]

SYMBOL TABLE

    BG_              78
    BLACK_           61
    BLINKBIT         52
    BLINKING         46
    BLUE_            62
    BROWN_           67
    Boolean_T       133     134
    COLORMODE        15      16      21      22
    CYAN_            64
    ClrEol          109
    ClrEop          110
    ClrScrn         106
    EXT_KBD          17
    FAR              16      17      19      20      21      23      25      26+
    FreeScrnBuf             126
    GRAY_            69
    GREEN_           63
    GetCurAtr       108
    GetCurPos       107
    GotoXY          105
    HIGHBLINK        51
    HIGHLITE         50
    HILTBIT          53
    HIWHITE_         76
    LTBLUE_          70
    LTCYAN_          72
    LTGREEN_         71
    LTMAGENTA_               74
    LTRED_           73
    MAGENTA_         66
    NORMAL           49
    RED_             65
    REVBLINK         48
    REVERSE          47
    ROWSIZE          19
    Repaint         111
    RestoreScrn             125
    SCANLINES        20
    SCRBUFF          21
    SCREEN          118     124     125     126
    SCREENCOLS               25
    SCREENROWS               26
    SCREENSEG        22
    SCRNBYTES        23      24
    SCRNMACS__H               9      10
    SCRNPIXELS               24
    SCROLL_DN        91
    SCROLL_UP        90
    SaveScrn        124
    VIDPAGE          18
    WHITE_           68
    YELLOW_          75
    a       78+
    col             105     107
    curX            120
    curY            121
    defined          15
    direction        93
    fRestoreScrn            134
    fSaveScrn       133
    far              18
    fname           133     134
    lrcolumn         99
    lrrow            98
    make_ansi       140
    num_lines        94
    row             105     107
    screen          125     126
    scroll           93
    ulcomumn         97
    ulrow            96
    vatr            140
    vattrib          95     106     111
    vbuf            119
```

```
/* +++Date last modified: 05-Jul-1997 */

/* program:    mousword.c
 * programmer: Ray L. McVay
 * date:       20 Oct 1988
 * modified:   15 Feb 93 by Bob Stout to use Bob Jarvis' MOUSE.H and MOUSE.C
 *
 * Demonstration of picking "words" off a text mode PC screen using a mouse.
 * Submitted to the C_ECHO and placed in the public domain, 7 Jun 1992.
 */

#include <stdio.h>
#include <stdlib.h>
#include "mouse.h"
#include "scrnmacs.h"

char word[80];
unsigned FAR *scrn;

void getword(char *, int, int);

main(void)
{
      int done, b, x, y;

      scrn = SCRBUFF;
      ms_reset(&b); /* reset */
      ms_show_cursor();
      for (done = 0; !done; )
      {
            b = ms_get_mouse_pos(&x, &y);
            if (b == 1)
            {
                  ms_hide_cursor();
                  getword(word, x/8, y/8);
                  do
                  {
                        b = ms_get_mouse_pos(&x, &y);
                  } while (b);
                  if (*word)
                        printf("{%s}\n", word);
                  ms_show_cursor();
            }
            else if (b > 1)
                  done = 1;
      }
      ms_reset(&b);
      return 0;
}

void getword(char *w, int x, int y)
{
      int txs, txe, ci;

      for (txs = x; (txs >= 0) && ((scrn[80 * y + txs] & 255) != 32); txs--)
            scrn[80 * y + txs] = (scrn[80 * y + txs] & 255) | 0x7000;
      for (txe = x; (txe < 80) && ((scrn[80 * y + txe] & 255) != 32); txe++)
            scrn[80 * y + txe] = (scrn[80 * y + txe] & 255) | 0x7000;
      for (ci = txs + 1; ci < txe; ci++)
            *w++ = (char)scrn[80 * y + ci];
      *w = 0;
}
```

TEXT STATISTICS

```
  1701 characters
    62 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    FAR             18
    SCRBUFF         26
    b       24      27      31      32      38      39      44      47
    ci              53      59+     60
    done            24      29+     45
    getword         20      35      51
    h       12      13
    main            22
    ms_get_mouse_pos        31      38
    ms_hide_cursor          34
    ms_reset        27      47
    ms_show_cursor          28      42
    printf          41
    scrn            18      26      55      56+     57      58+     60
    stdio           12
    stdlib          13
    txe             53      57+     58+     59
    txs             53      55+     56+     59
    w       51      60      61
    word            17      35      40      41
    x       24      31      35      38      51      55      57
    y       24      31      35      38      51      55      56+     57      58+     60
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Portable PC screen functions
**  Public domain by Bob Stout
**  Uses VIDPORT.C, also from SNIPPETS
**  Uses _fmemcpy(), available in most PC compiler libraries
**  A portable _fmemcpy() is available in FMEMOPS.C, also from SNIPPETS
*/

#include <stdlib.h>
 #if defined(__POWERC) || (defined(__ZTC__) && !defined(__SC__)) || \
       (defined(__TURBOC__) && !defined(__BORLANDC__))
  #include "fmemops.h"
 #else
  #include <string.h>
 #endif
#include "scrnmacs.h"         /* Also in SNIPPETS     */

/*
**  Save the text screen
*/

struct SCREEN *SaveScrn(void)
{
      struct SCREEN *screen;

      if (NULL == (screen = malloc(sizeof(struct SCREEN))))
            return NULL;
      if (NULL == (screen->vbuf = malloc(SCRNBYTES)))
      {
            free(screen);
            return NULL;
      }
      _fmemcpy((unsigned short FAR *)(screen->vbuf), SCRBUFF, SCRNBYTES);
      GetCurPos(&screen->curX, &screen->curY);
      return screen;
}

/*
**  Restore the text screen
*/

void RestoreScrn(struct SCREEN *screen)
{
      _fmemcpy(SCRBUFF, (unsigned short FAR *)(screen->vbuf), SCRNBYTES);
      GotoXY(screen->curX, screen->curY);
}

/*
**  Free a saved screen buffer
*/

void FreeScrnBuf(struct SCREEN *screen)
{
      free(screen->vbuf);
      free(screen);
}


#ifdef TEST

#include <stdio.h>
#include <conio.h>

/*
**  Run this test with a screenful of misc. stuff
*/

main()
{
      struct SCREEN *screen;
      int vatr = GetCurAtr();

      if (NULL == (screen = SaveScrn()))
      {
            puts("Unable to save the screen");
            return 1;
      }
      ClrScrn(vatr);
      GotoXY(0, 0);
      fputs("ClrScrn() tested", stderr);
      fputs("\nHit any key to continue...\n", stderr);
      getch();
      RestoreScrn(screen);
      FreeScrnBuf(screen);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1957 characters
    90 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     5 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    ClrScrn          80
    FAR              35      46
    FreeScrnBuf              54      86
    GetCurAtr        73
    GetCurPos        36
    GotoXY           47      81
    NULL             28      29      30      33      75
    RestoreScrn              44      85
    SCRBUFF          35      46
    SCREEN           24      26      28      44      54      72
    SCRNBYTES        30      35      46
    SaveScrn         24      75
    TEST             61
    __BORLANDC__             13
    __POWERC         12
    __SC__           12
    __TURBOC__               13
    __ZTC__          12
    _fmemcpy         35      46
    conio            64
    curX             36      47
    curY             36      47
    defined          12+     13+
    fputs            82      83
    free             32      56      57
    getch            84
    h       11       16      63      64
    main             70
    malloc           28      30
    puts             77
    screen           26      28      30      32      35      36+     37      44      46      47+     54
             56      57      72      75      85      86
    stderr           82      83
    stdio            63
    stdlib           11
    string           16
    vatr             73      80
    vbuf             30      35      46      56
```

```
/* +++Date last modified: 05-Jul-1997 */

/*--------------------------[ scroll ]--------------------------*/
/*      Scroll the active page up or down a number of lines     */
/*              Public domain code by Jeff Dunlop:              */
/*----------------------------------------------------------------*/
/* input:                                                         */
/*      dx = direction                                            */
/*      num_lines = number of lines to scroll, 0 = clear coords */
/*      attr = attribute of blank line(s)                         */
/*      y1, x1, y2, x2 = corner coordinates of scroll window    */
/* local:                                                         */
/*      regs = register union for ISR                             */
/*----------------------------------------------------------------*/

#include <dos.h>
#include "scrnmacs.h"

void scroll(int direction,
            int num_lines,
            int vattrib,
            int ulrow,
            int ulcomumn,
            int lrrow,
            int lrcolumn)
{
      union REGS regs;

      /*
            BH = attribute to be used on blank line
            CH = row of upper left corner of scroll window
            CL = column of upper left corner of scroll window
            DH = row of lower right corner of scroll window
            DL = column of lower right corner of scroll window
      */

      regs.h.al = (unsigned char)num_lines;
      regs.h.bh = (unsigned char)vattrib;
      regs.h.ch = (unsigned char)ulrow;
      regs.h.cl = (unsigned char)ulcomumn;
      regs.h.dh = (unsigned char)lrrow;
      regs.h.dl = (unsigned char)lrcolumn;

      if (direction == SCROLL_UP)
            regs.h.ah = 0x06;
      else  regs.h.ah = 0x07;

      int86(0x10, &regs, &regs);
}
```

TEXT STATISTICS

```
  1843 characters
    49 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    REGS            27
    SCROLL_UP       44
    ah              45      46
    al              37
    bh              38
    ch              39
    cl              40
    dh              41
    direction       19      44
    dl              42
    dos             16
    h       16      37      38      39      40      41      42      45      46
    int86           48
    lrcolumn        25      42
    lrrow           24      41
    num_lines       20      37
    regs            27      37      38      39      40      41      42      45      46      48+
    scroll          19
    ulcomumn        23      40
    ulrow           22      39
    vattrib         21      38
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPPETS string searching functions
  5 | */
  6 |
  7 | void  init_search(const char *string);                /* Pbmsrch.C      */
  8 | char *strsearch(const char *string);                  /* Pbmsrch.C      */
  9 | void  bmh_init(const char *pattern);                  /* Bmhsrch.C      */
 10 | char *bmh_search(const char *string,                  /* Bmhsrch.C      */
 11 |                  const int stringlen);
 12 | void  bmhi_init(const char *pattern);                 /* Bhmisrch.C     */
 13 | char *bmhi_search(const char *string,                 /* Bhmisrch.C     */
 14 |                   const int stringlen);
 15 | void  bmha_init(const char *pattern);                 /* Bmhasrch.C     */
 16 | char *bmha_search(const char *string,                 /* Bmhasrch.C     */
 17 |                   const int stringlen);
```

TEXT STATISTICS

```
   825 characters
    17 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    bmh_init         9
    bmh_search              10
    bmha_init       15
    bmha_search             16
    bmhi_init       12
    bmhi_search             13
    init_search              7
    pattern          9      12      15
    string           7       8      10      13      16
    stringlen       11      14      17
    strsearch        8
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SEQTOUCH.C - Touch files in a directory with sequential time stamps.
**
**  Public domain by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#ifdef __TURBOC__
 #include <dir.h>
#else
 #include <direct.h>
#endif
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif
#include "ftime.h"
#include "datetime.h"
#include "getopts.h"
#include "errors.h"
#include "snipfile.h"
#include "dosfiles.h"
#include "snipkbio.h"
#include "dirport.h"

/*
**  Options area - uses getopts() from SNIPPETS.
*/

char fname[FILENAME_MAX] = "";
char pname[FILENAME_MAX] = "";
char datestr[40];
char timestr[20];

Boolean_T help    = False_;
Boolean_T noquery = False_;

struct Option_Tag options[] = {
      {'F',  False_, String_Tag,  fname,    NULL, NULL, NULL},
      {'P',  False_, String_Tag,  pname,    NULL, NULL, NULL},
      {'D',  False_, String_Tag,  datestr,  NULL, NULL, NULL},
      {'T',  False_, String_Tag,  timestr,  NULL, NULL, NULL},
      {'H',  False_, Boolean_Tag, &help,    NULL, NULL, NULL},
      {'N',  False_, Boolean_Tag, &noquery, NULL, NULL, NULL},
      {'\0', False_, Error_Tag,   NULL,     NULL, NULL, NULL}
};

void loadtm(struct tm *ftm, struct ftime *ftimep)
{
      ftimep->ft_tsec  = ftm->tm_sec / 2;
      ftimep->ft_min   = ftm->tm_min;
      ftimep->ft_hour  = ftm->tm_hour;
      ftimep->ft_day   = ftm->tm_mday;
      ftimep->ft_month = ftm->tm_mon + 1;
      ftimep->ft_year  = ftm->tm_year - 80;
}

void usage(int errlvl)
{
      puts("SEQTOUCH - A utility to process a directory so its files "
           "have sequentially");
      puts("           increasing time/date stamps\n");
      puts("Usage: SEQTOUCH [-N] [-Ddir] [-Ffile] [-Dmm-dd-yy] "
           "[-Thh:mm:ss]\n");
      puts("Where: <dir> is the directory to process");
      puts("       <file> is an existing file whose time/date stamp "
           "to use");
      puts("       <mm-dd-yy> is a date in month-day-year format");
      puts("       <hh:mm:ss> is a time in hours:minutes:seconds "
           "format\n");
      puts("       <-N> suppresses verification of arguments\n");
      puts("All are optional. Specifying SEQTOUCH with no arguments "
           "causes all files in");
      puts("the current directory to be dated to the current date. "
           "The timestamp of the");
      puts("first file will be set to the current date and time and "
           "each subsequent file");
      puts("will have its time/date stamp set to 2 seconds later "
           "than the preceding file.\n");
      puts("Specifying the -T and/or -D options sets the time and/or date "
           "of the first");
      puts("file in the directory to the speified time and/or date.\n");
      puts("Specifying a file with the -F option causes the timestamp of "
           "first file in ");
      puts("the directory to be set to match the specified file.\n");
      puts("If the -N option is not specified, you will br prompted with "
           "the specific");
      puts("directory, time, and date (and file, if specified) to be used.");
      puts("The option switches are not case sensitive, i.e. -f works just "
           "like -F.");
      exit(errlvl);
}

int main(int argc, char *argv[])
```

```
{
      struct ftime ftimep;
      struct tm ftm;
      time_t ft;
      DOSFileData ff;
      char str[50];
      char path[FILENAME_MAX];
      char file[FILENAME_MAX];
      size_t plen;

      if (Error_ == getopts(argc, argv))
            usage(EXIT_FAILURE);

      if (help)
            usage(EXIT_SUCCESS);

      if (NUL != pname[0])
      {
            if (!isdir(pname))
                  ErrExit("%s is not a directory", pname);
      }
      else  getcwd(pname, FILENAME_MAX);

      if (NUL != fname[0])
      {
            FILE *fp;

            fp = cant(fname, "r");
            if (Success_ != getftime(fileno(fp), &ftimep))
                  ErrExit("Can't read timestamp of %s", fname);
            ftime2tm(&ftimep, &ftm);
            ft = ftime2time(&ftimep);
      }
      else
      {
            time(&ft);
            ftm = *localtime(&ft);

            if (NUL != datestr[0])
            {
                  unsigned yy, mm, dd;

                  if (Success_ != parse_date(datestr, &yy, &mm, &dd, USA))
                        ErrExit("Invalid date - %s", datestr);
                  ftm.tm_year = yy - 1900;
                  ftm.tm_mon  = mm - 1;
                  ftm.tm_mday = dd;
            }

            if (NUL != timestr[0])
            {
                  unsigned  hh, mm, ss;

                  if (Success_ != parse_time(timestr, &hh, &mm, &ss))
                        ErrExit("Invalid time - %s", timestr);
                  ftm.tm_hour = hh;
                  ftm.tm_min  = mm;
                  ftm.tm_sec  = ss;
            }

            loadtm(&ftm, &ftimep);
      }

      printf("\nDirectory %s\n", pname);
      if (NUL == fname[0])
            puts("No file specified\n");
      else  printf("Using file %s\n\n", fname);
      strftime(str, 50, "%A, %d-%b-%Y, %X", &ftm);
      printf("Files will be timstamped sequentially beginning with\n  %s\n",
             str);
      if (!noquery)
      {
            if (False_ == getYN("OK?", 'N', 5))
            {
                  fputs("\nSEQTOUCH aborted by user\n", stderr);
                  return EXIT_SUCCESS;
            }
      }

      strcpy(path, pname);
      if ('\\' != LAST_CHAR(path))
            strcat(path, "\\");
      plen = strlen(path);
      strcpy(file, path);

      strcat(path, "*.*");
      if (Success_ == FIND_FIRST(path, _A_NORMAL, &ff)) do
      {
            FILE *fp;

            file[plen] = NUL;
            strcat(file, ff_name(&ff));

            fprintf(stderr, "Touching %s\n", file);

            fp = cant(file, "r+");
            setftime(fileno(fp), &ftimep);
            fclose(fp);

            ftm.tm_sec += 2;
```

```
 201 |             ft = mktime(&ftm);            /* Used to normalize time     */
 202 |             loadtm(&ftm, &ftimep);
 203 |       } while (Success_ == FIND_NEXT(&ff));
 204 |
 205 |       return EXIT_SUCCESS;
 206 | }
```

TEXT STATISTICS

```
  6376 characters
   206 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    22 preprocessor instructions
     9 constants [character]
    58 constants [string]
    16 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       41      42
    Boolean_Tag             49      50
    DOSFileData             105
    EXIT_FAILURE            112
    EXIT_SUCCESS            115     176     205
    ErrExit         120     130     144     155
    Error_          111
    Error_Tag        51
    FILE            126     189
    FILENAME_MAX             36      37     107     108     122
    FIND_FIRST              187
    FIND_NEXT       203
    False_           41      42      45      46      47      48      49      50      51     173
    LAST_CHAR       181
    MSDOS            18
    NUL             117     124     139     150     165     191
    NULL             45+     46+     47+     48+     49+     50+     51+
    Option_Tag               44
    String_Tag               45      46      47      48
    Success_        129     143     154     187     203
    USA             143
    _A_NORMAL       187
    __MSDOS__        18
    __TURBOC__               13
    argc            100     111
    argv            100     111
    cant            128     196
    conio            11
    datestr          38      47     139     143     144
    dd              141     143     147
    defined          18+
    dir              14
    direct           16
    errlvl           64      97
    exit             97
    fclose          198
    ff              105     187     192     203
    ff_name         192
    file            108     184     191     192     194     196
    fileno          129     197
    fname            36      45     124     128     130     165     167
    fp              126     128     129     189     196     197     198
    fprintf         194
    fputs           175
    ft              104     132     136     137     201
    ft_day           59
    ft_hour          58
    ft_min           57
    ft_month         60
    ft_tsec          56
    ft_year          61
    ftime            54     102
    ftime2time              132
    ftime2tm        131
    ftimep           54      56      57      58      59      60      61     102     129     131     132
           161      197     202
    ftm              54      56      57      58      59      60      61     103     131     137     145
           146      147     156     157     158     161     168     200     201     202
    getYN           173
    getcwd          122
    getftime        129
    getopts         111
    h          9     10      11      12      14      16      21
    help             41      49     114
    hh              152     154     156
    isdir           119
    loadtm           54     161     202
    localtime       137
    main            100
    mktime          201
    mm              141     143     146     152     154     157
    noquery          42      50     171
    options          44
    parse_date              143
    parse_time              154
    path            107     180     181     182     183     184     186     187
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
+--------------------------------------------------+
|              Thunderbird Software                |
+--------------------------------------------------+
| Filespec  :  Serial.c                            |
| Date      :  October 24, 1991                    |
| Time      :  15:03                               |
| Revision  :  1.1                                 |
|     Update: August 29, 1994                      |
+--------------------------------------------------+
| Programmer:  Scott Andrews                       |
| Address   :  5358 Summit RD SW                   |
| City/State:  Pataskala, Ohio                     |
| Zip       :  43062                               |
+--------------------------------------------------+
| Released to the Public Domain                    |
+--------------------------------------------------+
*/

/*
+----------------------------------------------------------+
|  Call open_serial to install the interrupt handler       |
|  You must call close_serial before exiting your program  |
|  or a machine crash will occur!                          |
+----------------------------------------------------------+
*/

#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include "serial.h"
#include "queue.h"

QUEUE *Serial_In_Queue;
QUEUE *Serial_Out_Queue;

OLD_COMM_PARAMS old_comm_params;
COMM_STATUS     comm_status;

void (INTERRUPT FAR *oldvector_serial )();
/* save addr for intr handler */

int  ComBase;                       /* Comm port address */
int  IrqNum;                        /* Comm interrupt request */

int OpenComPort ( char Port )       /* install int. handler */
{
      unsigned status;
      int retval = -1;

      /* allocate input and output queues */

      Serial_In_Queue = alloc_queue( SerInBufSize );
      if ( (QUEUE *) 0 == Serial_In_Queue)
            return retval;
      Serial_Out_Queue = alloc_queue( SerOutBufSize );
      if ( (QUEUE *) 0 == Serial_Out_Queue)
      {
            free ( Serial_In_Queue );
            return retval;
      }
      retval = 0;

      /* Setup Comm base port address and IRQ number */

      switch ( Port)
      {
            case '1': ComBase = 0x3F8; IrqNum = 4; break;
            case '2': ComBase = 0x2F8; IrqNum = 3; break;
            case '3': ComBase = 0x3E8; IrqNum = 4; break;
            case '4': ComBase = 0x2E8; IrqNum = 3; break;
            default : ComBase = 0x3F8; IrqNum = 4; break;
      }
      old_comm_params.int_enable = inp ( ComBase + INT_EN );
      outp ( ComBase + INT_EN, 0 ); /* turn off comm interrupts */

      /* save old comm parameters */

      old_comm_params.line = inp ( ComBase + LINE_CNTRL );
      old_comm_params.modem = inp ( ComBase + MODEM_CNTRL );
      status = inp ( ComBase + LINE_CNTRL );
      outp ( ComBase + LINE_CNTRL, (unsigned char) status | 0x80 );
      old_comm_params.baud_lsb = inp ( ComBase + BAUD_LSB );
      old_comm_params.baud_msb = inp ( ComBase + BAUD_MSB );
      status = inp ( ComBase + LINE_CNTRL );
      outp ( ComBase + LINE_CNTRL, (unsigned char) status | 0x7F );
      status = OUT2 | DTR;              /* DTR/OUT2 must be set! */
      outp ( ComBase + MODEM_CNTRL, (unsigned char) status );

      /* get serial port address/vector */

      oldvector_serial = (void(INTERRUPT FAR *)(void))getvect(IrqNum + 8 );

      /* set our interrupt handler */

      setvect ( IrqNum + 8, serial );

      /* save the PIC */
```

```
101 |
102 |       old_comm_params.int_cntrl = inp ( 0x21 );
103 |       status = ( 1 << IrqNum);      /* calculate int enable bit */
104 |       status = ~status;
105 |
106 |       /* ok enable comm ints */
107 |
108 |       outp ( 0x21, (unsigned char) old_comm_params.int_cntrl &
109 |             (unsigned char) status );
110 |       return retval;
111 | }
112 |
113 | void CloseComPort ( void )
114 | {
115 |       int status;
116 |
117 |       /* restore UART to previous state */
118 |
119 |       outp ( ComBase + INT_EN, (unsigned char) 0 );
120 |       outp ( ComBase + MODEM_CNTRL,
121 |             (unsigned char) old_comm_params.modem );
122 |       status = inp ( ComBase + LINE_CNTRL );
123 |       outp ( ComBase + LINE_CNTRL,
124 |             (unsigned char) status | 0x80 );
125 |       outp ( ComBase + BAUD_LSB,
126 |             (unsigned char) old_comm_params.baud_lsb );
127 |       outp ( ComBase + BAUD_MSB,
128 |             (unsigned char) old_comm_params.baud_msb );
129 |       outp ( ComBase + LINE_CNTRL,
130 |             (unsigned char) old_comm_params.line );
131 |       outp ( 0x21, (unsigned char) old_comm_params.int_cntrl );
132 |
133 |       /* restore old interrupt handler */
134 |
135 |       setvect ( IrqNum + 8, oldvector_serial );
136 |
137 |       /* free input and output queues */
138 |
139 |       free ( Serial_In_Queue );
140 |       free ( Serial_Out_Queue );
141 |       return;
142 | }
143 |
144 | void InitComPort ( char Baud[], char Databits,
145 |                    char Parity, char Stopbits )
146 | {
147 |       int status;
148 |       long baudrate;
149 |       unsigned divisor;
150 |
151 |       /* set baud rate */
152 |
153 |       status = inp ( ComBase + LINE_CNTRL );
154 |       outp ( ComBase + LINE_CNTRL,
155 |             (unsigned char) status | 0x80 );
156 |       baudrate = atol ( Baud );
157 |       if ( baudrate == 0)
158 |             baudrate = 2400L;
159 |       divisor = (unsigned) ( 115200L / baudrate);
160 |       outp ( ComBase + BAUD_LSB,
161 |             (unsigned char) ( divisor & 0x00FF) );
162 |       outp ( ComBase + BAUD_MSB,
163 |             (unsigned char) ( ( divisor >> 8) & 0x00FF) );
164 |       status = 0x00;
165 |
166 |       /* set parity */
167 |
168 |       switch ( Parity)                    /* set parity value     */
169 |       {
170 |       case 'O':                           /* odd parity           */
171 |       case 'o':
172 |             status = 0x08; break;
173 |
174 |       case 'E':                           /* even parity          */
175 |       case 'e':
176 |             status = 0x18; break;
177 |
178 |       case 'S':                           /* stick parity         */
179 |       case 's':
180 |             status = 0x28; break;
181 |
182 |       case 'N':                           /* no parity            */
183 |       case 'n':
184 |             default :   status = 0x00;
185 |       }
186 |
187 |     /* set number data bits */
188 |
189 |       switch ( Databits)
190 |       {
191 |       case '5':
192 |             break;
193 |
194 |       case '6':
195 |             status = status | 0x01;
196 |             break;
197 |
198 |       case '7':
199 |             status = status | 0x02;
200 |             break;
```

```
201 |
202 |       case '8':
203 |       default :
204 |             status = status | 0x03;
205 |       }
206 |
207 |       /* set number stop bits */
208 |
209 |       switch ( Stopbits)
210 |       {
211 |       case '2':
212 |             status = status | 0x04;
213 |             break;
214 |
215 |       case '1':
216 |       default :
217 |             ;
218 |       }
219 |       outp ( ComBase + LINE_CNTRL, (unsigned char) status );
220 |       status = OUT2 | DTR;        /* DTR/OUT2 must be set! */
221 |       outp ( ComBase + MODEM_CNTRL, (unsigned char) status );
222 |
223 |       /* enable serial interrupts */
224 |
225 |       outp ( ComBase + INT_EN, RX_INT | ERR_INT | RS_INT );
226 |       return;
227 | }
228 |
229 | void DropDtr ( void )
230 | {
231 |       int status;
232 |
233 |       status = inp ( ComBase + MODEM_CNTRL );
234 |       status &= 0xFE;                /* turn off DTR bit */
235 |       outp ( ComBase + MODEM_CNTRL, (unsigned char) status );
236 |       return;
237 | }
238 |
239 | void RaiseDtr ( void )
240 | {
241 |       int status;
242 |
243 |       status = inp ( ComBase + MODEM_CNTRL );
244 |       status |= 0x01;                /* turn on DTR bit */
245 |       outp ( ComBase + MODEM_CNTRL, (unsigned char) status );
246 |       return;
247 | }
248 |
249 | int ComRecChar ( void )
250 | {
251 |       return de_queue ( Serial_In_Queue );
252 | }
253 |
254 | int ComSendString ( char *string )
255 | {
256 |       int retval;
257 |       char *pointer;
258 |       pointer = string;
259 |
260 |       while ( *pointer)
261 |       {
262 |             retval = en_queue ( Serial_Out_Queue, *pointer );
263 |             pointer++;
264 |       }
265 |       if ( 0x0 == (comm_status.modem & 0x40))
266 |             RaiseDtr ();
267 |       outp ( ComBase + INT_EN, RX_INT | TBE_INT | ERR_INT | RS_INT );
268 |       return retval;
269 | }
270 |
271 | int ComSendChar ( char character )
272 | {
273 |       int retval;
274 |
275 |       /* interrupt driven send */
276 |
277 |       if ( 0x0 == (comm_status.modem & 0x40))
278 |             RaiseDtr ();
279 |       retval = en_queue ( Serial_Out_Queue, character );
280 |       if ( - 1 != retval)
281 |             outp ( ComBase + INT_EN, RX_INT | TBE_INT | ERR_INT | RS_INT );
282 |       return retval;
283 | }
284 |
285 | int ComStatus ( void )
286 | {
287 |       unsigned status;
288 |       unsigned retval;
289 |
290 |       retval = inp ( ComBase + LINE_STATUS );
291 |       retval = retval << 8;
292 |       status = inp ( ComBase + MODEM_STATUS );
293 |       retval = retval | status;
294 |       if ( queue_empty ( Serial_In_Queue ))
295 |             retval &= 0xFEFF;
296 |       else  retval |= 0x0100;
297 |       return (int) retval;
298 | }
299 |
300 | void INTERRUPT FAR serial ( void )           /* interrupt handler */
```

```
{
      int temp;

      disable ();
      while ( 1)
      {
            comm_status.intrupt = inp ( ComBase + INT_ID );
            comm_status.intrupt &= 0x0f;
            switch ( comm_status.intrupt)
            {
            case 0x00:                  /* modem interrupt */
                  comm_status.modem = inp( ComBase + MODEM_STATUS );
                  break;

            case 0x02:                  /* xmit interrupt */
                  if ( queue_empty ( Serial_Out_Queue ))
                        outp(ComBase + INT_EN, RX_INT|ERR_INT|RS_INT );
                  else
                  {
                        temp = de_queue ( Serial_Out_Queue );
                        if ( - 1 != temp)
                              outp ( ComBase + XMIT, temp );
                  }
                  break;

            case 0x04:                  /* receive interrupt */
                  en_queue(Serial_In_Queue, (char)inp(ComBase + REC));
                  break;

            case 0x06:                  /* line interrupt */
                  comm_status.line = inp ( ComBase + LINE_STATUS );
                  (void) inp ( ComBase + REC );
                  en_queue ( Serial_In_Queue, '!' );
                  break;

            default:                    /* No Mo` Left */
                  comm_status.modem = inp ( ComBase + MODEM_STATUS );
                  outp ( 0x20, 0x20 );
                  enable ();
                  return;
            }   /* switch */
      }   /* while */
}
/* End of Serial.C */
```

TEXT STATISTICS

10357 characters
344 lines

LEXICAL STATISTICS

43 comments [std-C]
0 comments [C++]
5 preprocessor instructions
19 constants [character]
2 constants [string]
61 constants [numeric]

SYMBOL TABLE

```
BAUD_LSB          85     125     160
BAUD_MSB          86     127     162
Baud             144     156
COMM_STATUS       40
CloseComPort     113
ComBase           45      70      71      72      73      74      76      77      81      82      83
          84      85      86      87      88      90     119     120     122     123     125     127
         129     153     154     160     162     219     221     225     233     235     243     245
         267     281     290     292     307     312     317     322     327     331     332     337
ComRecChar       249
ComSendChar      271
ComSendString    254
ComStatus        285
DTR               89     220
Databits         144     189
DropDtr          229
ERR_INT          225     267     281
FAR               42      94     300
INTERRUPT         42      94     300
INT_EN            76      77     119     225     267     281     317
INT_ID           307
InitComPort      144
IrqNum            46      70      71      72      73      74      94      98     103     135
LINE_CNTRL        81      83      84      87      88     122     123     129     153     154
         219
LINE_STATUS      290     331
MODEM_CNTRL       82      90     120     221     233     235     243     245
MODEM_STATUS     292     312     337
OLD_COMM_PARAMS   39
OUT2              89     220
OpenComPort       48
Parity           145     168
Port              48      68
QUEUE             36      37      56      59
REC              327     332
RS_INT           225     267     281
RX_INT           225     267     281
RX_INT|ERR_INT|RS_INT    317
RaiseDtr         239     266     278
SerInBufSize      55
SerOutBufSize     58
Serial_In_Queue   36      55      56      61     139     251     294     327     333
Serial_Out_Queue  37      58      59     140     262     279     316     320
Stopbits         145     209
TBE_INT          267     281
XMIT             322
alloc_queue       55      58
atol             156
baud_lsb          85     126
baud_msb          86     128
baudrate         148     156     157     158     159
character        271     279
comm_status       40     265     277     307     308     309     312     331     337
de_queue         251     320
disable          304
divisor          149     159     161     163
dos               31
en_queue         262     279     327     333
enable           339
free              61     139     140
getvect           94
h        30       31      32
inp               76      81      82      83      85      86      87     102     122     153     233
         243     290     292     307     312     327     331     332     337
int_cntrl        102     108     131
int_enable        76
intrupt          307     308     309
line              81     130     331
modem             82     121     265     277     312     337
old_comm_params   39      76      81      82      85      86     102     108     121     126
         128     130     131
oldvector_serial  42      94     135
outp              77      84      88      90     108     119     120     123     125     127     129
         131     154     160     162     219     221     225     235     245     267     281     317
         322     338
pointer          257     258     260     262     263
queue_empty      294     316
retval            51      57      62      64     110     256     262     268     273     279     280
         282     288     290     291+   293+    295     296     297
serial            98     300
setvect           98     135
status            50      83      84      87      88      89      90     103     104+    109     115
         122     124     147     153     155     164     172     176     180     184     195+    199+
         204+    212+    219     220     221     231     233     234     235     241     243     244
```

```
        245     287     292     293
stdlib           30
string           32     254     258
temp            302     320     321     322
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
+--------------------------------------------------+
|              Thunderbird Software                |
+--------------------------------------------------+
| Filespec  :  Serial.c                            |
| Date      :  October 24, 1991                    |
| Time      :  15:03                               |
| Revision  :  1.1                                 |
| Update    : August 29, 1994                      |
| Update    : March 12, 1995 by Bob Stout          |
+--------------------------------------------------+
| Programmer:  Scott Andrews                       |
| Address   :  5358 Summit RD SW                   |
| City/State:  Pataskala, Ohio                     |
| Zip       :  43062                               |
+--------------------------------------------------+
| Released to the Public Domain                    |
+--------------------------------------------------+
*/

#ifndef SERIAL__H
#define SERIAL__H

#include "extkword.h"
#include "pchwio.h"

#define  SerInBufSize  4096           /* Size of input buffer          */
#define  SerOutBufSize 512            /* Size of output buffer         */

/* 8250 registers */

#define  REC             0            /* Uart receive reg.             */
#define  XMIT            0            /* Uart transmit reg.            */
#define  INT_EN          1            /* Uart int. enable reg.         */
#define  INT_ID          2            /* Uart int. ident. reg.         */
#define  LINE_CNTRL      3            /* Uart line control reg.        */
#define  MODEM_CNTRL     4            /* Uart modem control reg.       */
#define  LINE_STATUS     5            /* Uart line status reg.         */
#define  MODEM_STATUS    6            /* Uart modem status reg.        */
#define  BAUD_LSB        0            /* Uart baud divisor reg.        */
#define  BAUD_MSB        1            /* Uart baud divisor reg.        */

#define  NONE            0            /* Handshake param none          */
#define  HDW             1            /* Handshake param hardware      */
#define  XON             2            /* Handshake param software      */

/* Interrupt enable register  */

#define  RX_INT          0x01         /* Receive interrupt mask        */
#define  TBE_INT         0x02         /* Transmit buffer empty mask    */
#define  ERR_INT         0x04         /* Error interrupt mask          */
#define  RS_INT          0x08         /* Line interrupt mask           */

/* Interrupt id register      */

#define  OUT2            0x08         /* Out 2 line                    */
#define  DTR             0x01         /* DTR high                      */
#define  RTS             0x02         /* RTS high                      */
#define  CTS             0x10
#define  DSR             0x20
#define  XMTRDY          0x20
#define  TXR             0       /*  Transmit register (WRITE)        */

#if !defined TRUE                     /* Define boolean true/false     */
#define  FALSE  0
#define  TRUE   !FALSE
#endif

extern void (INTERRUPT FAR *oldvector_serial )( void);

extern int   ComBase;                 /* Comm port address             */
extern int   IrqNum;                  /* Comm interrupt request        */

typedef struct                        /* Save existing comm params     */
{  int int_enable;                    /* old interrupt enable reg value*/
   int line;                          /*  "  line control      "    "  */
   int modem;                         /* old modem control     "    "  */
   int baud_lsb;                      /* old baud rate divisor LSD     */
   int baud_msb;                      /*  "   "    "        "    MSD     */
   int int_cntrl;                     /* old PIC interrupt reg value   */
} OLD_COMM_PARAMS;
extern OLD_COMM_PARAMS old_comm_params;

typedef struct
{  int line;                          /* Uart line status reg.         */
   int modem;                         /* Uart mode status reg.         */
   int intrupt;                       /* Uart interrupt reg.           */
   int handshake;                     /* Handshake status              */
} COMM_STATUS;                        /* status, updated, handler      */
extern COMM_STATUS comm_status;

int   OpenComPort ( char Port );         /*setup comm for usage         */
void  InitComPort ( char Baud[], char Databits, char Parity, char Stop );
void  CloseComPort ( void );             /* Restore comm port           */
void  DropDtr ( void );                  /* Lower DTR                   */
void  RaiseDtr ( void );                 /* Raise DTR                   */
int   ComRecChar ( void );               /* Fetch character from rcv buf*/

```

```
 101 | int   ComSendChar ( char character );    /* Put char into xmit buffer   */
 102 | int   ComSendString ( char *string );
 103 | int   ComStatus ( void );                /* Fetch comm status           */
 104 | void  INTERRUPT FAR serial ( void );     /* interrupt handler           */
 105 |
 106 | /* End of Serial.H */
 107 |
 108 | #endif /* SERIAL__H */


TEXT STATISTICS

  5181 characters
   108 lines

LEXICAL STATISTICS

    53 comments [std-C]
     0 comments [C++]
    35 preprocessor instructions
     0 constants [character]
     2 constants [string]
    27 constants [numeric]

SYMBOL TABLE

    BAUD_LSB        42
    BAUD_MSB        43
    Baud            95
    COMM_STATUS              91      92
    CTS             61
    CloseComPort             96
    ComBase         73
    ComRecChar               99
    ComSendChar             101
    ComSendString           102
    ComStatus      103
    DSR              62
    DTR              59
    Databits         95
    DropDtr          97
    ERR_INT          53
    FALSE            67      68
    FAR              71     104
    HDW              46
    INTERRUPT        71     104
    INT_EN           36
    INT_ID           37
    InitComPort              95
    IrqNum           74
    LINE_CNTRL               38
    LINE_STATUS              40
    MODEM_CNTRL              39
    MODEM_STATUS             41
    NONE             45
    OLD_COMM_PARAMS          83      84
    OUT2             58
    OpenComPort              94
    Parity           95
    Port             94
    REC              34
    RS_INT           54
    RTS              60
    RX_INT           51
    RaiseDtr         98
    SERIAL__H        23      24
    SerInBufSize             29
    SerOutBufSize            30
    Stop             95
    TBE_INT          52
    TRUE             66      68
    TXR              64
    XMIT             35
    XMTRDY           63
    XON              47
    baud_lsb         80
    baud_msb         81
    character       101
    comm_status              92
    defined          66
    handshake        90
    int_cntrl        82
    int_enable               77
    intrupt          89
    line             78      87
    modem            79      88
    old_comm_params          84
    oldvector_serial         71
    serial          104
    string          102
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SETENVAR.C - Program which sets the DOS master environment upon exit
  5 | **
  6 | **  Original Copyright 1988-1991 by Bob Stout as part of
  7 | **  the MicroFirm Function Library (MFL)
  8 | **
  9 | **  The user is granted a free limited license to use this source file
 10 | **  to create royalty-free programs, subject to the terms of the
 11 | **  license restrictions specified in the LICENSE.MFL file.
 12 | */
 13 |
 14 | #include <stdio.h>
 15 | #include <string.h>
 16 | #include <stdlib.h>
 17 | #include <conio.h>
 18 | #include "sniptype.h"
 19 | #include "errors.h"                /* For cant()                        */
 20 | #include "snipkbio.h"              /* For ungetkey() & KB_stuff()       */
 21 |
 22 | main(int argc, char *argv[])
 23 | {
 24 |       FILE *bfile;
 25 |
 26 |       if (3 > argc)
 27 |       {
 28 |             puts("\aUsage: SETENVAR envar datum");
 29 |             abort();
 30 |       }
 31 |       bfile = cant("$TMP$.BAT", "w");
 32 |       fprintf(bfile, "SET %s=%s\ndel $tmp$.bat\x1a", argv[1], argv[2]);
 33 |       fclose(bfile);
 34 |       while (kbhit())
 35 |             getch();
 36 |       KB_stuff("$tmp$\r");
 37 |       return 0;
 38 | }
```

TEXT STATISTICS

```
  1089 characters
    38 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     8 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    FILE            24
    KB_stuff        36
    abort           29
    argc            22      26
    argv            22      32+
    bfile           24      31      32      33
    cant            31
    conio           17
    fclose          33
    fprintf         32
    getch           35
    h       14      15      16      17
    kbhit           34
    main            22
    puts            28
    stdio           14
    stdlib          16
    string          15
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SETIMETO.C - Set the timestamp of one file to match another.
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include <stdlib.h>
 11 | #include <dos.h>
 12 | #include <io.h>
 13 | #include <fcntl.h>
 14 | #include "ftime.h"
 15 |
 16 | int main(int argc, char *argv[])
 17 | {
 18 |       int fd0, fd1;
 19 |
 20 |       struct ftime Ftime;
 21 |
 22 |       if (3 > argc)
 23 |       {
 24 |             puts("Usage: SETIMETO filename_w/original_stamp "
 25 |                   "filename_w/stamp_to_set");
 26 |             return EXIT_FAILURE;
 27 |       }
 28 |
 29 |       if (-1 == (fd0 = open(argv[1], O_RDONLY)))
 30 |       {
 31 |             printf("Unable to open %s\n", argv[1]);
 32 |             return EXIT_FAILURE;
 33 |       }
 34 |
 35 |       getftime(fd0, &Ftime);             /* Save the time/date         */
 36 |
 37 |       if (-1 == (fd1 = open(argv[2], O_WRONLY)))
 38 |       {
 39 |             printf("Unable to open %s\n", argv[2]);
 40 |             return EXIT_FAILURE;
 41 |       }
 42 |
 43 |       setftime(fd1, &Ftime);             /* Set the time/date          */
 44 |
 45 |       close(fd0);
 46 |       close(fd1);
 47 |       return EXIT_SUCCESS;
 48 | }
```

TEXT STATISTICS

```
  1092 characters
    48 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     5 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            26      32      40
    EXIT_SUCCESS            47
    Ftime           20      35      43
    O_RDONLY        29
    O_WRONLY        37
    argc            16      22
    argv            16      29      31      37      39
    close           45      46
    dos             11
    fcntl           13
    fd0             18      29      35      45
    fd1             18      37      43      46
    ftime           20
    getftime        35
    h       9       10      11      12      13
    io              12
    main            16
    open            29      37
    printf          31      39
    puts            24
    setftime        43
    stdio            9
    stdlib          10
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************
 *                          SETLEVEL                             *
 *                                                               *
 * Written by: Lynn R. Lively                                    *
 * Date:       05/15/90                                          *
 * Version:    1.0                                               *
 * Written in TurboC V2.00                                       *
 *                                                               *
 * Allows user input in a MSDOS .BAT file. This program asks a   *
 * specified question and sets the errorlevel according to valid *
 * answer list. Since MSDOS provides no facility for getting user*
 * input into a .BAT file, and manipulation of enviorinment strings *
 * seems to be unreliable and nonportable, I have written this   *
 * facility as a possible alternative.                           *
 *                                                               *
 * I hereby place this program into the public domain.           *
 * Users of this utility do so at their own risk. I accept no    *
 * responsibility for the accuracy or useability of this software.  *
 * However, should you have any questions, suggestions, or problems *
 * I would be happy to talk with you (and help if I can).        *
 * My current work number is 713-591-6111                        *
 *  Your Servant,                                                *
 *       Lynn R. Lively                                          *
 *                                                               *
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

/*
 * The following are bit mask values for the option_flg.
 */

#define CASE_SENSE       1
#define SINGLE_KEY_ENTRY 2

/*
 * The following is the structure deffinition for the valid answer
 * linked list.
 */

typedef struct v_ans {
      char *         ans_key;
      int            rtn_err_lev;
      struct v_ans * next_ans;
} VALID_ANS;

/*
 * Prototypes of functions defined in this module.
 */

void usage (char * progname);

int main (int argc, char * argv[])
{
      register int i = 0;
      register int j;

      int option_flg = 0;

      VALID_ANS * answer_tab = NULL;
      VALID_ANS * wk_ans;

      char * wk_ptr;
      char * ques_ptr = "?";
      char   wk_str[256];

      if (argc > 1)
      {
        /*
         * Step through and parse the argument list.
         */

            for (i = 1; i < argc; i++)
            {
                  j = 0;
                  switch (argv[i][j])
                  {
                  case '/':            /* Options are flagged by '/' */

                        j++;
                        switch (argv[i][j])
                        {
                        case 'c':
                        case 'C':

                              option_flg |= CASE_SENSE;
                              break;

                        case '1':

                              option_flg |= SINGLE_KEY_ENTRY;
                              break;

                        case 'q':
                        case 'Q':
```

```
101 |
102 |                         /*
103 |                          * The question is assumed to be the argument
104 |                          * after the 'q'.
105 |                          */
106 |
107 |                               ques_ptr = strdup (argv[++i]);
108 |                               break;
109 |
110 |                         default:
111 |
112 |                         /*
113 |                          * Oops! Something wrong. Tell them how its
114 |                          * supposed to work.
115 |                          */
116 |
117 |                               usage (argv[0]);
118 |                               return (-1);
119 |                         }
120 |                         break;
121 |
122 |                   case '-':         /* Answers are flagged by '-' */
123 |
124 |                         /*
125 |                          * The answer is assumed to be everything from the
126 |                          * '-' to the '=' characters. This allows for
127 |                          * single letter and word answers.
128 |                          */
129 |
130 |                         j++;
131 |                         if ((wk_ptr = strtok (argv[i]+j, "=")) != NULL)
132 |                         {
133 |                               /*
134 |                                * Allocate space for the new answer entry and
135 |                                * link in into the list. Answers are linked
136 |                                * into the list in reverse order from their
137 |                                * command line specification since the logic
138 |                                * is considerably simpler and it shouldn't
139 |                                * matter much anyway.
140 |                                */
141 |
142 |                               if (answer_tab == NULL)
143 |                               {
144 |                                     answer_tab = wk_ans = (VALID_ANS *)
145 |                                           calloc (1, sizeof(VALID_ANS));
146 |                               }
147 |                               else
148 |                               {
149 |                                     wk_ans = (VALID_ANS *)
150 |                                                 calloc (1, sizeof(VALID_ANS));
151 |                                     wk_ans->next_ans = answer_tab;
152 |                                     answer_tab = wk_ans;
153 |                               }
154 |
155 |                               /*
156 |                                * Store the answer string and errorlevel value
157 |                                * into the answer element.
158 |                                */
159 |
160 |                               wk_ans->ans_key = strdup (wk_ptr);
161 |                               wk_ptr          = strtok (NULL, "=");
162 |                               if (wk_ptr != NULL)
163 |                               {
164 |                                     /*
165 |                                      * Allow only positive returns since -1 is
166 |                                      * and error return from the program.
167 |                                      */
168 |
169 |                                     wk_ans->rtn_err_lev =
170 |                                           abs (atoi (wk_ptr));
171 |                               }
172 |                               else  wk_ans->rtn_err_lev = 0;
173 |                         }
174 |                         }
175 |                         break;
176 |
177 |                   default:
178 |
179 |                   /*
180 |                    * Oops! Something wrong. Tell them how its
181 |                    * supposed to work.
182 |                    */
183 |
184 |                         usage (argv[0]);
185 |                         return (-1);
186 |             }
187 |       }
188 |       else
189 |       {
190 |
191 |             /*
192 |              * Oops! Something wrong. Tell them how its
193 |              * supposed to work.
194 |              */
195 |
196 |             usage (argv[0]);
197 |             return (-1);
198 |       }
199 |
200 |       /*
```

```
201 |        * Ask the question and get their answer.
202 |        */
203 |
204 |       printf ("%s", ques_ptr);
205 |
206 |       if ((option_flg & SINGLE_KEY_ENTRY) != 0)
207 |       {
208 |             wk_str[0] = getche ();
209 |             wk_str[1] = '\0';
210 |       }
211 |       else  fgets (wk_str, sizeof (wk_str), stdin);
212 |
213 |       /*
214 |        * See if we can find their answer in our valid answer list.
215 |        */
216 |
217 |       wk_ans = answer_tab;
218 |       while (wk_ans != (VALID_ANS *) NULL)
219 |       {
220 |             if ((option_flg & CASE_SENSE) != 0)
221 |             {
222 |                   if ((strncmp (wk_str, wk_ans->ans_key,
223 |                         (strlen (wk_ans->ans_key)))) == 0)
224 |                   {
225 |                   /*
226 |                    * If we found the answer return the associated
227 |                    * errorlevel return number.
228 |                    */
229 |
230 |                         return (wk_ans->rtn_err_lev);
231 |                   }
232 |             }
233 |             else
234 |             {
235 |                   if ((strnicmp (wk_str, wk_ans->ans_key,
236 |                         (strlen (wk_ans->ans_key)))) == 0)
237 |                   {
238 |                   /*
239 |                    * If we found the answer return the associated
240 |                    * errorlevel return number.
241 |                    */
242 |
243 |                         return (wk_ans->rtn_err_lev);
244 |                   }
245 |             }
246 |
247 |             wk_ans = wk_ans->next_ans;
248 |       }
249 |
250 |       /*
251 |        * If the answer wasn't in our table return a -1.
252 |        */
253 |
254 |       return (-1);
255 | }
256 |
257 | void usage (char * progname)
258 | {
259 |       printf ("%s usage:\n", progname);
260 |       printf ("%s /[options] -<valid answer>=<errorlevel>\n\n", progname);
261 |       printf ("Where:\n");
262 |       printf ("  options      = c    (Answers are case sensitive)\n");
263 |       printf ("                 1    (Single key entry)\n");
264 |       printf ("                 q\"Question Text\"\n\n");
265 |       printf ("  valid answer = (Single letter or word response)\n");
266 |       printf ("  errorlevel   = (errorlevel to return when answer seen)\n\n");
267 |       printf ("Example:\n%s /q\"Answer Y or N: \" /1 -y=1 -n=2\n\n",
268 |             progname);
269 |       printf ("Note: -1 is returned if answer is not valid or syntax\n");
270 |       printf ("      is incorrect.\n");
271 | }
```

TEXT STATISTICS

```
  9439 characters
   271 lines
```

LEXICAL STATISTICS

```
    21 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     8 constants [character]
    15 constants [string]
    24 constants [numeric]
```

SYMBOL TABLE

```
    CASE_SENSE              38      91     220
    NULL            65     131     142     161     162     218
    SINGLE_KEY_ENTRY        39      96     206
    VALID_ANS       50      65      66     144     145     149     150     218
    abs            170
    ans_key         47     160     222     223     235     236
    answer_tab              65     142     144     151     152     217
    argc            58      72      78
    argv            58      81      86     107     117     131     184     196
    atoi           170
    calloc         145     150
    conio           32
    fgets          211
    getche         208
    h       29      30      31      32
    i       60      78+     81      86     107     131
    j       61      80      81      85      86     130     131
    main            58
    next_ans        49     151     247
    option_flg              63      91      96     206     220
    printf         204     259     260     261     262     263     264     265     266     267     269
           270
    progname        56     257     259     260     268
    ques_ptr        69     107     204
    rtn_err_lev             48     169     172     230     243
    stdin          211
    stdio           29
    stdlib          30
    strdup         107     160
    string          31
    strlen         223     236
    strncmp        222
    strnicmp       235
    strtok         131     161
    usage           56     117     184     196     257
    v_ans           46      49
    wk_ans          66     144     149     151     152     160     169     172     217     218     222
           223     230     235     236     243     247+
    wk_ptr          68     131     160     161     162     170
    wk_str          70     208     209     211+    222     235
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SETVOL.C - set, change, or kill a disk volume label
**
**  public domain demo by Bob Stout
**  DOS 5 enhancements suggested by Keith Beedle
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>
#include <io.h>
#include "sniptype.h"
#include "dirport.h"
#include "snpdskio.h"
#include "dosfiles.h"
#include "mk_fp.h"

#if defined(__TURBOC__)
 #pragma option -a-
#else
 #if defined(__ZTC__)
  #pragma ZTC align 1
 #else /* MSC/QC/WATCOM/METAWARE */
  #pragma pack(1)
 #endif
 struct fcb_ {
         char   fcb_drive;
         char   fcb_name[8];
         char   fcb_ext[3];
         short  fcb_curblk;
         short  fcb_recsize;
         long   fcb_filsize;
         short  fcb_date;
         char   fcb_resv[10];
         char   fcb_currec;
         long   fcb_random;
 };

 struct xfcb_ {
         char           xfcb_flag;
         char           xfcb_resv[5];
         char           xfcb_attr;
         struct fcb_    xfcb_fcb;
 };

 #define fcb      fcb_
 #define xfcb     xfcb_
#endif

#include "dos5boot.h"   /* SNIPPETS file with DOS 5 boot record structure  */

/*
**  Erase an existing volume label
*/

void vol_kill(char *fname)
{
      union REGS regs;
      struct SREGS sregs;
      struct xfcb buf;

      /* Parse the filename into an FCB              */

      segread(&sregs);
      regs.h.ah = 0x29;
      regs.h.al = 0;
      regs.x.si = (unsigned)fname;
      regs.x.di = (unsigned)&buf.xfcb_fcb;
      sregs.es  = sregs.ds;
      intdosx(&regs, &regs, &sregs);

      /* Volume labels require extended FCB's        */

      buf.xfcb_flag = 0xff;
      buf.xfcb_attr = _A_VOLID;

      /* Delete the old label                        */

      regs.h.ah = 0x13;
      regs.x.dx = (unsigned)&buf;
      intdos(&regs, &regs);
}

/*
**  Create a new volume label
*/

void setvol(char *label)
{
      char new_label[13];     /* name + ext + '.' + NUL       */
      struct xfcb buf;
      union REGS regs;
      struct SREGS sregs;
      const char pattern[] = "????????";
      char FAR *dta;

```

```
101 |       /*
102 |       **  Change to root directory.
103 |       */
104 |
105 |       PushDir("\\");
106 |
107 |       /* If drive is already labeled, remove it               */
108 |
109 |       segread(&sregs);
110 |       regs.h.ah = 0x2f;
111 |       intdosx(&regs, &regs, &sregs);
112 |       dta = MK_FP(sregs.es, regs.x.bx);
113 |
114 |       buf.xfcb_flag = 0xff;
115 |       buf.xfcb_attr = _A_VOLID;
116 |       buf.xfcb_fcb.fcb_drive = 0;
117 |       memcpy(buf.xfcb_fcb.fcb_name, pattern, 8);
118 |       memcpy(buf.xfcb_fcb.fcb_ext, pattern, 3);
119 |
120 |       regs.h.ah = 0x11;
121 |       regs.x.dx = (unsigned)&buf;
122 |       intdos(&regs, &regs);
123 |
124 |       if (0 == regs.h.al)
125 |       {
126 |             int i;
127 |             char oldlabel[13], FAR *p, *q;
128 |
129 |             for (i = 0, p = dta + 8, q =oldlabel; i < 8; ++i, ++p, ++q)
130 |             {
131 |                   *q = *p;
132 |             }
133 |             *q++ = '.';
134 |             for (i = 0, p = dta + 16; i < 3; ++i, ++p, ++q)
135 |             {
136 |                   *q = *p;
137 |             }
138 |             vol_kill(oldlabel);
139 |       }
140 |
141 |       strcpy(new_label, label);
142 |       if (8 < strlen(label))
143 |       {
144 |             new_label[8] = '.';
145 |             strcpy(&new_label[9], &label[8]);
146 |       }
147 |
148 |       /* Parse the filename into an FCB               */
149 |
150 |       segread(&sregs);
151 |       regs.h.ah = 0x29;
152 |       regs.h.al = 0;
153 |       regs.x.si = (unsigned)new_label;
154 |       regs.x.di = (unsigned)&buf.xfcb_fcb;
155 |       sregs.es  = sregs.ds;
156 |       intdosx(&regs, &regs, &sregs);
157 |
158 |       /* Volume labels require extended FCB's         */
159 |
160 |       buf.xfcb_flag = 0xff;
161 |       buf.xfcb_attr = _A_VOLID;
162 |
163 |       /* Create the new label                         */
164 |
165 |       regs.h.ah = 0x16;
166 |       regs.x.dx = (unsigned)&buf;
167 |       intdos(&regs, &regs);
168 |
169 |       /* Close the new label                          */
170 |
171 |       regs.h.ah = 0x10;
172 |       regs.x.dx = (unsigned)&buf;
173 |       intdos(&regs, &regs);
174 |
175 |       /*
176 |       **  For DOS 5.0 replace the boot record too.
177 |       */
178 |
179 |       if(_osmajor > 3)
180 |       {
181 |             int index, drive = getdrv();
182 |             B_REC boot_record;
183 |
184 |             AbsDiskRead(drive, 1, 0, &boot_record);
185 |             if(0 == strcmp(boot_record.bsOemName, "MSDOS5.0"))
186 |             {
187 |                   index = 0;
188 |                   while (NUL != label[index])
189 |                   {
190 |                         boot_record.bsVolumeLabel[index] = label[index];
191 |                         index++;
192 |                   }
193 |                   for( ; index < 11; index++)
194 |                         boot_record.bsVolumeLabel[index] = 0x20;
195 |                   AbsDiskWrite(drive, 1, 0, &boot_record);
196 |             }
197 |       }
198 |       PopDir();
199 | }
200 |
```

```
 201 | #ifdef TEST
 202 |
 203 | main(int argc, char *argv[])
 204 | {
 205 |       if (2 > argc)
 206 |       {
 207 |             puts("\aUsage: SETVOL new_name");
 208 |             abort();
 209 |       }
 210 |       setvol(argv[1]);
 211 |       return EXIT_SUCCESS;
 212 | }
 213 |
 214 | #endif /* TEST */
```

TEXT STATISTICS

```
  5079 characters
   214 lines
```

LEXICAL STATISTICS

```
    18 comments [std-C]
     0 comments [C++]
    25 preprocessor instructions
     2 constants [character]
    10 constants [string]
    45 constants [numeric]
```

SYMBOL TABLE

```
    AbsDiskRead            184
    AbsDiskWrite           195
    B_REC          182
    EXIT_SUCCESS           211
    FAR             99     127
    MK_FP          112
    NUL            188
    PopDir         198
    PushDir        105
    REGS            62      96
    SREGS           63      97
    TEST           201
    ZTC             26
    _A_VOLID        79     115     161
    __TURBOC__              22
    __ZTC__         25
    _osmajor       179
    a       23
    abort          208
    ah              69      83     110     120     151     165     171
    al              70     124     152
    align           26
    argc           203     205
    argv           203     210
    boot_record            182     184     185     190     194     195
    bsOemName      185
    bsVolumeLabel          190     194
    buf             64      72      78      79      84      95     114     115     116     117     118
          121      154     160     161     166     172
    bx             112
    ctype           13
    defined         22      25
    di              72     154
    dos             14
    drive          181     184     195
    ds              73     155
    dta             99     112     129     134
    dx              84     121     166     172
    es              73     112     155
    fcb             50
    fcb_            30      47      50
    fcb_curblk              34
    fcb_currec              39
    fcb_date        37
    fcb_drive       31     116
    fcb_ext         33     118
    fcb_filsize             36
    fcb_name        32     117
    fcb_random              40
    fcb_recsize             35
    fcb_resv        38
    fname           60      71
    getdrv         181
    h       10      11      12      13      14      15      69      70      83     110     120     124
          151     152     165     171
    i      126     129+    134+
    index          181     187     188     190+    191     193+    194
    intdos          85     122     167     173
    intdosx         74     111     156
    io              15
    label           92     141     142     145     188     190
    main           203
    memcpy         117     118
    new_label       94     141     144     145     153
    oldlabel       127     129     138
    option          23
    p      127     129+    131     134+    136
    pack            28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SHEL2DOS.C - Shell to DOS from a running program
**
**  Original Copyright 1989-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include "snpdosys.h"

int shell_to_DOS(void)
{
      char *comspec, prompt[256], *oldprompt;
      int retval;

      comspec = getenv("COMSPEC");
      if(comspec == NULL)
            comspec = "COMMAND.COM";     /* Better than nothing... */

      sprintf(prompt, "PROMPT=[Type EXIT to return to program]\r\n%s",
            oldprompt = getenv("PROMPT"));
      putenv(prompt);

      retval = spawnlp(0, comspec, comspec, NULL);

      sprintf(prompt, "PROMPT=%s", oldprompt);
      putenv(prompt);

      return retval;
}

#ifdef TEST

#include <stdio.h>

main()
{
      int retval = shell_to_DOS();

      printf("shell_to_DOS() returned %d\n", retval);

      retval = shell_to_DOS();
      printf("shell_to_DOS() returned %d\n", retval);
      return 0;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1322 characters
    56 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     8 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    NULL            26      33
    TEST            41
    comspec         22      25      26      27      33+
    getenv          25      30
    h       14      15      16      17      43
    main            45
    oldprompt       22      30      35
    printf          49      52
    process         17
    prompt          22      29      31      35      36
    putenv          31      36
    retval          23      33      38      47      49      51      52
    shell_to_DOS            20      47      51
    spawnlp         33
    sprintf         29      35
    stdio           14      43
    stdlib          15
    string          16
```

```
/* +++Date last modified: 05-Jul-1997 */

/* - skiplist.c ------------------------------------------------- *
**
** Once upon a time Skiplist were invented by William Pugh at the
** University of Maryland (according to Bruce Schneier, DrDobbs,
** january 1994). They are an elegant alternative approach to balanced
** binary threes.
**
** Unfortunately the implementation shown in that same article did
** not yield the expected results, so I decided to "roll my own"
** rather than looking for errors in somebody elses code.
**
** The result (surprise?) looked remarkably like the original version,
** except that it worked with my compiler, that is, untill I turned on
** the optimizer! (Lattice 5.06.02)
**
** I then started to tweak the code, changing while to do/while or
** variable++; to ++variable; in order to fool the optimizer into
** doing what I intended to be done. (All very esoteric ;)
**
** I have since then tested with gcc (DecStation) which works, and
** also cc -O3 (DecStation) which now works (This last one one was
** a beast!!)
**
** If your compiler/optimizer still cannot digest this,
** then please let me know,
**
** else tweak this file to be useful in your own projects
** and (if you like?) send me a "postcard" :-)
**
** 17.08.1996 // Jens M Andreasen <jens-and@dsv.su.se>
**
** In a message of 20-Aug-1996, Jens M Andreasen wrote:
**
** My skiplist.c listing is hereby explicitly given the status of freeware.
** Distribute as is with the following additions ...
**
** ------------------------------------------------------------------------
**
** Note that the integers used for key and value are assumed to be 32 bit.
**
** Note also that the randomgenerator is a temporary hack. The original
** implementation has a better one. And do take a look at the alternative
** approches to DUPLICATEFOUND while you are at it:
*/

#include<limits.h>
#include<stdlib.h>
#include<stdio.h>

#define LEVELMAX 15
#define keytype int
#define valuetype int
#define KEYMAX INT_MAX

typedef struct structnode node;
typedef struct structnode{
      int key;
      int value;
      node *next[1];
} *pt_node;


/* ------------------------------------------------------------------ */

#define New(type) malloc(sizeof (type))
#define NewArray(type,x) malloc(sizeof(type) * (x))
#define NewNode(randomlevel)\
 malloc(sizeof(node) + (randomlevel) * sizeof(pt_node))

int insertnode(node **List, int key, int value);
#define DUPLICATEFOUND /* return -1 */

int findnode(node **List, int key);
#define INITVALUE 0

int deletenode(node **List, int key);

void findall(node **List);/* under construction */

node **newlist(void);

int main(void);


/* ------------------------------------------------------------------ */



int insertnode(node **List, int key, int value)
{
      int r,level = LEVELMAX;
      node *temp  = List[0], **bTrack[LEVELMAX+1];

      if (temp->key > key)
      {
            do
                  bTrack[level] = List;
            while (--level >= 0);
```

```
101 |       }
102 |       else
103 |       {
104 |             do
105 |             {
106 |                   while((temp = List[level]), (temp->key < key))
107 |                   {
108 |                         List = temp->next;
109 |                   }
110 |                   bTrack[level] = List;
111 |             } while (--level >= 0);
112 |
113 |             if (temp->key == key)
114 |             {
115 |                   DUPLICATEFOUND;
116 |                   temp->value = value;
117 |                   return 0;
118 |             }
119 |       }
120 |
121 |       r = rand(); /* not so random, but works for now ... */
122 |       level = 0;
123 |
124 |       while (r & 0x4000)
125 |       {
126 |             r <<= 1;
127 |             ++level;
128 |       }
129 |
130 |       temp = NewNode(level);
131 |       if (!temp)
132 |       {
133 |             printf(" Out of memory at key: %d",key);
134 |             exit(1);
135 |       }
136 |
137 |       temp->value = value;
138 |       temp->key   = key;
139 |
140 |       do{
141 |             temp->next[level] = bTrack[level][level];
142 |             bTrack[level][level] = temp;
143 |       } while (--level >= 0);
144 |
145 |       return 1;
146 |  }
147 |
148 | int findnode(node **List, int key)
149 | {
150 |       int  level = LEVELMAX;
151 |       node *temp = List[0];
152 |
153 |       if (temp->key > key)
154 |             return INITVALUE;
155 |
156 |       do
157 |       {
158 |             while(temp = List[level], (temp->key < key))
159 |             {
160 |                   List = temp->next;
161 |             }
162 |       } while (--level >= 0);
163 |
164 |       if (temp->key == key)
165 |       {
166 |             return temp->value;
167 |       }
168 |       else  return INITVALUE;
169 |  }
170 |
171 |
172 |
173 | int deletenode(node **List, int key)
174 | {
175 |       int level  = LEVELMAX;
176 |       node *temp = List[0], **bTrack[LEVELMAX + 1];
177 |
178 |       if ((temp->key > key)||(KEYMAX == key))
179 |             return 0;
180 |
181 |       do
182 |       {
183 |             while(bTrack[level] = List, temp = List[level], (temp->key < key))
184 |             {
185 |                   List=temp->next;
186 |             }
187 |
188 |       } while (--level >= 0);
189 |
190 |       if (temp->key == key)
191 |       {
192 |             level=0;
193 |             do
194 |             {
195 |                   bTrack[level][level] = temp->next[level];
196 |             } while (++level, bTrack[level][level]->key == key);
197 |
198 |             free(temp);
199 |             return -1;
200 |       }
```

```
201 |       return 0;
202 |  }
203 |
204 |
205 |
206 | void findall(node **List)                /* just testing ... */
207 | {
208 |       while (List && List[0])
209 |       {
210 |             ++List[0]->value;
211 |             --List[0]->value;
212 |
213 |             List = List[0]->next;
214 |       }
215 | }
216 |
217 | node **newlist(void)
218 | {
219 |       int l = LEVELMAX;
220 |       node *NIL,
221 |
222 |       **L = NewArray(pt_node, LEVELMAX + 1);
223 |
224 |       NIL = NewNode(LEVELMAX);
225 |
226 |       NIL->key   = KEYMAX;
227 |       NIL->value = INITVALUE;
228 |       do
229 |       {
230 |             NIL->next[l] = NULL;
231 |       } while (l--);
232 |
233 |       l=LEVELMAX;
234 |       do
235 |       {
236 |             L[l] = NIL;
237 |       } while (l--);
238 |
239 |       return L;
240 |  }
241 |
242 |
243 | /* ---------------------------------------------------------------- */
244 |
245 |
246 | int main(void)
247 | {
248 |       int i;
249 |       node **L;
250 |
251 |       if (sizeof(int) < 4)
252 |       {
253 |             puts("This code must be compiled with 32-bit ints!");
254 |             return EXIT_FAILURE;
255 |       }
256 |
257 |       L=newlist();
258 |
259 |       puts(" DOWN");
260 |
261 |       for(i = 100000; i >= 0; i -=2)
262 |             insertnode(L, i, i);
263 |
264 |       puts(" UP");
265 |       for(i = 1; i < 100000; i += 2)
266 |             insertnode(L, i, i);
267 |
268 |       deletenode(L, 40);
269 |
270 |       puts(" FIND");
271 |       for(i = -2; i <= 100002; ++i)
272 |             findnode(L, i);
273 |
274 |       puts(" FAST");
275 |       findall(L);
276 |
277 |       puts("SAMPLES");
278 |       printf(" %d", findnode(L, -10));
279 |
280 |       printf(" %d,", findnode(L, 0));
281 |       printf(" %d", findnode(L, 1));
282 |       printf(" %d", findnode(L, 2));
283 |       printf(" %d", findnode(L, 39));
284 |       printf(" %d", findnode(L, 40));
285 |       printf(" %d", findnode(L, 41));
286 |       printf(" %d", findnode(L, 42));
287 |       printf(" %d", findnode(L, 99999));
288 |       printf(" %d", findnode(L, 100000));
289 |
290 |       printf(", %d", findnode(L, 100001));
291 |       printf(" %d", findnode(L, 100008));
292 |
293 |       puts(" DONE");
294 |
295 |       return EXIT_SUCCESS;
296 | }
```

TEXT STATISTICS

```
  7103 characters
   296 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
    12 preprocessor instructions
     0 constants [character]
    20 constants [string]
    50 constants [numeric]
```

SYMBOL TABLE

```
    DUPLICATEFOUND          73     115
    EXIT_FAILURE           254
    EXIT_SUCCESS           295
    INITVALUE       76     154     168     227
    INT_MAX         55
    KEYMAX          55     178     226
    L      222     236     239     249     257     262     266     268     272     275     278     280
           281     282     283     284     285     286     287     288     290     291
    LEVELMAX        52      93      94     150     175     176     219     222     224     233
    List            72      75      78      80      91      94      99     106     108     110     148
           151     158     160     173     176     183+    185     206     208+    210     211     213+
    NIL            220     224     226     227     230     236
    NULL           230
    New             67
    NewArray        68     222
    NewNode         69     130     224
    bTrack          94      99     110     141     142     176     183     195     196
    deletenode              78     173     268
    exit           134
    findall         80     206     275
    findnode        75     148     272     278     280     281     282     283     284     285     286
           287     288     290     291
    free           198
    h       48      49      50
    i      248     261+    262+    265+    266+    271+    272
    insertnode              72      91     262     266
    key             59      72      75      78      91      96+    106+    113+    133     138+    148
           153+    158+    164+    173     178+    183+    190+    196+    226
    keytype         53
    l      219     230     231     233     236     237
    level           93      99     100     106     110     111     122     127     130     141+    142+
           143     150     158     162     175     183+    188     192     195+    196+
    limits          48
    main            84     246
    malloc          67      68      70
    newlist         82     217     257
    next            61     108     141     160     185     195     213     230
    node            57      61      70      72      75      78      80      82      91      94     148
           151     173     176     206     217     220     249
    printf         133     278     280     281     282     283     284     285     286     287     288
           290     291
    pt_node         62      70     222
    puts           253     259     264     270     274     277     293
    r       93     121     124     126
    rand           121
    randomlevel             69      70
    stdio           50
    stdlib          49
    structnode              57      58
    temp            94      96     106+    108     113     116     130     131     137     138     141
           142     151     153     158+    160     164     166     176     178     183+    185     190
           195     198
    type            67+     68+
    value           60      72      91     116+    137+    166     210     211     227
    valuetype       54
    x       68+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Header file for portable file functions
  5 | */
  6 |
  7 | #ifndef SNIPFILE__H
  8 | #define SNIPFILE__H
  9 |
 10 | #include <stdio.h>
 11 | #include "sniptype.h"
 12 |
 13 | #define flushall() fflush(NULL)
 14 |
 15 | long   flength(char *fname);                          /* Ansiflen.C     */
 16 | FILE * cant(char *fname, char *fmode);                /* Ferrorf.C      */
 17 | int    fcompare(const char *fnam1, const char *fnam2);/* Fcompare.C     */
 18 | long   fcopy(char *dest, char *source);               /* Fcopy.C        */
 19 | long   ffsearch(FILE *fp, const char *pattern,
 20 |              const size_t size, int N);              /* Srchfile.C     */
 21 | long   rfsearch(FILE *fp, const char *pattern,
 22 |              const size_t size, int N);              /* Srchfile.C     */
 23 | void   show_text_file(char *txt);                    /* Textmod.C      */
 24 | int    file_copy(char *from, char *to);              /* Wb_Fcopy.C     */
 25 | int    fdcopy(int fdfrom, int fdto);                 /* Wb_Fcopy.C     */
 26 | int    file_append(char *from, char *to);            /* Wb_Fapnd.C     */
 27 |
 28 | Boolean_T exists(char *name);                        /* Existsx.C      */
 29 | char     *dexists(char *name, char *envar);          /* Existsx.C      */
 30 | char     *pexists(char *name);                       /* Existsx.C      */
 31 | char     *gexists(char *name, char *envar);          /* Existsx.C      */
 32 | FILE     *fopenp(char *name, char *mode);            /* Fopenx.C       */
 33 | FILE     *fopend(char *name, char *mode, char *envar);/* Fopenx.C       */
 34 | FILE     *fopeng(char *name, char *mode, char *envar);/* Fopenx.C       */
 35 |
 36 |
 37 | #endif /* SNIPFILE__H */
```

TEXT STATISTICS

```
  1643 characters
    37 lines
```

LEXICAL STATISTICS

```
    20 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       28
    FILE            16      19      21      32      33      34
    N       20      22
    NULL            13
    SNIPFILE__H              7       8
    cant            16
    dest            18
    dexists         29
    envar           29      31      33      34
    exists          28
    fcompare        17
    fcopy           18
    fdcopy          25
    fdfrom          25
    fdto            25
    fflush          13
    ffsearch        19
    file_append             26
    file_copy       24
    flength         15
    flushall        13
    fmode           16
    fnam1           17
    fnam2           17
    fname           15      16
    fopend          33
    fopeng          34
    fopenp          32
    fp              19      21
    from            24      26
    gexists         31
    h       10
    mode            32      33      34
    name            28      29      30      31      32      33      34
    pattern         19      21
    pexists         30
    rfsearch        21
    show_text_file          23
    size            20      22
    size_t          20      22
    source          18
    stdio           10
    to              24      26
    txt             23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**    SNIPKBIO.H - Snippets header file for keyboard I/O function
*/

#ifndef SNIPKBIO__H
#define SNIPKBIO__H

#include <dos.h>
#include <conio.h>
#include "sniptype.h"
#include "pchwio.h"

Boolean_T getYN(char *prompt, int def_ch,             /* Getyn.C       */
                unsigned timeout);

int  ungetkey(unsigned short key);                    /* Kb_Stuff.C    */
int  KB_stuff(char *str);                             /* Kb_Stuff.C    */

int  IsLeftShift(void);                               /* Isshift.C     */
int  IsRightShift(void);                              /* Isshift.C     */
int  IsShift(void);                                   /* Isshift.C     */

int  IsLeftAlt(void);                                 /* Isshift.C     */
int  IsRightAlt(void);                                /* Isshift.C     */
int  IsAlt(void);                                     /* Isshift.C     */

int  IsLeftCtl(void);                                 /* Isshift.C     */
int  IsRightCtl(void);                                /* Isshift.C     */
int  IsCtl(void);                                     /* Isshift.C     */

int  IsSysRq(void);                                   /* Isshift.C     */
int  timed_getch(int n_seconds);                      /* Timegetc.C    */
int  isxkeybrd(void);                                 /* Isxkbrd.C     */
void setcaps(void);                                   /* Keylocks.C    */
void clrcaps(void);                                   /* Keylocks.C    */
void setnumlock(void);                                /* Keylocks.C    */
void clrnumlock(void);                                /* Keylocks.C    */

#define     RIGHT_SHIFT       0x0001
#define     LEFT_SHIFT        0x0002

#define     EITHER_ALT        0x0008
#define     LEFT_ALT          0x0200

#define     EITHER_CTL        0x0004
#define     LEFT_CTL          0x0100

#if defined(__OS2__)
 #define    SYSRQ             0x8000
#else /* assume DOS */
 #define    SYSRQ             0x0400
#endif

#ifdef __OS2__
 #define INCL_NOPM
 #define INCL_KBD
 #define INCL_DOSPROCESS    /* for DosSleep() */
 #include <os2.h>

KBDINFO setkbmode(void);    /* Change keyboard to binary mode */
void restkbmode(KBDINFO);   /* restore keyboard mode */
                            /* both defined in EXT_KEYS.C */

/* 30-Mar-96 - EBB:
** OS/2 doesn't have a place in memory where information about the last
** key is held, like DOS does.      All the information about a keystroke is
** wrapped up into a structure (KBDKEYINFO) returned from OS/2's
** get-a-key API call, KbdCharIn(), so if you want to query that information
** at some arbitrary later time, you have to save it somewhere.  I have
** chosen to use a global variable defined in ISSHIFT.C to do this.  The
** functions in ISSHIFT.C report the status of the last key stored in that
** global, which may not be the last key pressed in your program if you're
** also storing keystrokes elsewhere.
*/
extern KBDKEYINFO ki;         /* Holds key info - defined in ISSHIFT.C */

 #define peekkey()      (&ki.fsState)
#else /* !__OS2__ */
 #define key_seg  0x40
 #define key_off  0x17
 #define peekkey()      ((unsigned short FAR*) MK_FP(key_seg, key_off))
#endif

#endif /* SNIPKBIO__H */
```

```
TEXT STATISTICS

  3373 characters
    86 lines

LEXICAL STATISTICS

    30 comments [std-C]
     0 comments [C++]
    29 preprocessor instructions
     0 constants [character]
     2 constants [string]
    10 constants [numeric]

SYMBOL TABLE

    Boolean_T       15
    EITHER_ALT              44
    EITHER_CTL              47
    FAR             83
    INCL_DOSPROCESS         59
    INCL_KBD        58
    INCL_NOPM       57
    IsAlt           27
    IsCtl           31
    IsLeftAlt       25
    IsLeftCtl       29
    IsLeftShift             21
    IsRightAlt              26
    IsRightCtl              30
    IsRightShift            22
    IsShift         23
    IsSysRq         33
    KBDINFO         62      63
    KBDKEYINFO              77
    KB_stuff        19
    LEFT_ALT        45
    LEFT_CTL        48
    LEFT_SHIFT              42
    MK_FP           83
    RIGHT_SHIFT             41
    SNIPKBIO__H              7       8
    SYSRQ           51      53
    __OS2__         50      56
    clrcaps         37
    clrnumlock              39
    conio           11
    def_ch          15
    defined         50
    dos             10
    fsState         79
    getYN           15
    h       10      11      60
    isxkeybrd       35
    key             18
    key_off         82      83
    key_seg         81      83
    ki              77      79
    n_seconds       34
    os2             60
    peekkey         79      83
    prompt          15
    restkbmode              63
    setcaps         36
    setkbmode       62
    setnumlock              38
    str             19
    timed_getch             34
    timeout         16
    ungetkey        18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SNIPMATH.H - Header file for SNIPPETS math functions and macros
*/

#ifndef SNIPMATH__H
#define SNIPMATH__H

#include <math.h>
#include "sniptype.h"
#include "round.h"

/*
**  Callable library functions begin here
*/

void    SetBCDLen(int n);                               /* Bcdl.C         */
long    BCDtoLong(char *BCDNum);                        /* Bcdl.C         */
void    LongtoBCD(long num, char BCDNum[]);             /* Bcdl.C         */
double  bcd_to_double(void *buf, size_t len,            /* Bcdd.C         */
                      int digits);
int     double_to_bcd(double arg, char *buf,            /* Bcdd.C         */
                      size_t length, size_t digits );
DWORD   ncomb1 (int n, int m);                          /* Combin.C       */
DWORD   ncomb2 (int n, int m);                          /* Combin.C       */
void    SolveCubic(double a, double b, double c,        /* Cubic.C        */
                   double d, int *solutions,
                   double *x);
DWORD   dbl2ulong(double t);                            /* Dbl2Long.C     */
long    dbl2long(double t);                             /* Dbl2Long.C     */
double  dround(double x);                               /* Dblround.C     */

/* Use #defines for Permutations and Combinations     -- Factoryl.C     */

#define log10P(n,r) (log10factorial(n)-log10factorial((n)-(r)))
#define log10C(n,r) (log10P((n),(r))-log10factorial(r))

double  log10factorial(double N);                       /* Factoryl.C     */

double  fibo(unsigned short term);                      /* Fibo.C         */
double  frandom(int n);                                 /* Frand.C        */
double  ipow(double x, int n);                          /* Ipow.C         */
int     ispow2(int x);                                  /* Ispow2.C       */
long    double ldfloor(long double a);                  /* Ldfloor.C      */
int     initlogscale(long dmax, long rmax);             /* Logscale.C     */
long    logscale(long d);                               /* Logscale.C     */

float   MSBINToIEEE(float f);                           /* Msb2Ieee.C     */
float   IEEEToMSBIN(float f);                           /* Msb2Ieee.C     */
int     perm_index (char pit[], int size);              /* Perm_Idx.C     */
int     round_div(int n, int d);                        /* Rnd_Div.C      */
long    round_ldiv(long n, long d);                     /* Rnd_Div.C      */
double  rad2deg(double rad);                            /* Rad2Deg.C      */
double  deg2rad(double deg);                            /* Rad2Deg.C      */

#include "pi.h"
#ifndef PHI
 #define PHI      ((1.0+sqrt(5.0))/2.0)         /* the golden number    */
 #define INV_PHI  (1.0/PHI)                     /* the golden ratio     */
#endif

/*
**  File: ISQRT.C
*/

struct int_sqrt {
      unsigned sqrt,
               frac;
};

void usqrt(unsigned long x, struct int_sqrt *q);


#endif /* SNIPMATH__H */
```

TEXT STATISTICS

```
  3007 characters
    75 lines
```

LEXICAL STATISTICS

```
    34 comments [std-C]
     0 comments [C++]
    13 preprocessor instructions
     0 constants [character]
     3 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    BCDNum          19      20
    BCDtoLong       19
    DWORD           25      26      30
    IEEEToMSBIN             50
    INV_PHI         60
    LongtoBCD       20
    MSBINToIEEE             49
    N       39
    PHI             58      59      60
    SNIPMATH__H              7       8
    SetBCDLen       18
    SolveCubic              27
    a       27      45
    arg             23
    b       27
    bcd_to_double           21
    buf             21      23
    c       27
    d       28      47      52      53
    dbl2long        31
    dbl2ulong       30
    deg             55
    deg2rad         55
    digits          22      24
    dmax            46
    double_to_bcd           23
    dround          32
    f       49      50
    fibo            41
    frac            69
    frandom         42
    h       10
    initlogscale            46
    int_sqrt        67      72
    ipow            43
    ispow2          44
    ldfloor         45
    len             21
    length          24
    log10C          37
    log10P          36      37
    log10factorial          36+     37      39
    logscale        47
    m       25      26
    math            10
    n       18      25      26      36+     37+     42      43      52      53
    ncomb1          25
    ncomb2          26
    num             20
    perm_index              51
    pit             51
    q       72
    r       36+     37+
    rad             54
    rad2deg         54
    rmax            46
    round_div       52
    round_ldiv              53
    size            51
    size_t          21      24+
    solutions       28
    sqrt            68
    t       30      31
    term            41
    usqrt           72
    x       29      32      43      44      72
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Header file for SNIPPETS PC printer control functions
*/

#ifndef SNIPRINT__H
#define SNIPRINT__H

#include <stdio.h>            /* For FILE */

/*
**  ASSIGNPRN.C
*/

#define NUM_OF_PRNTRS 6

extern FILE *printer[NUM_OF_PRNTRS];

int assign_printer(int number, char *device);

/*
**  CHANGPRN.C
*/

typedef enum {LPT1, LPT2, LPT3, COM1, COM2, CON} PrintDevice;

int change_prn(PrintDevice device);

/*
**  PRTOGGLE.C
*/

int prtoggle(void);

/*
**  PRTSCRN.C
*/

int PrtScrnStat(void);
int PrtScrn(void);

/*
**  PRTSTAT.C
*/

struct PrStatus {
      unsigned int timeout  : 1;
      unsigned int unused   : 2;
      unsigned int IOerror  : 1;
      unsigned int selected : 1;
      unsigned int paperout : 1;
      unsigned int ack      : 1;
      unsigned int notbusy  : 1;
};

int prtstat(unsigned int);


#endif /* SNIPRINT__H */
```

TEXT STATISTICS

944 characters
60 lines

LEXICAL STATISTICS

9 comments [std-C]
0 comments [C++]
5 preprocessor instructions
0 constants [character]
0 constants [string]
8 constants [numeric]

SYMBOL TABLE

COM1	26		
COM2	26		
CON	26		
FILE	18		
IOerror	50		
LPT1	26		
LPT2	26		
LPT3	26		
NUM_OF_PRNTRS		16	18
PrStatus	47		
PrintDevice		26	28
PrtScrn	41		
PrtScrnStat		40	
SNIPRINT__H		7	8
ack	53		
assign_printer		20	
change_prn		28	
device	20	28	
h 10			
notbusy	54		
number	20		
paperout	52		
printer	18		
prtoggle	34		
prtstat	57		
selected	51		
stdio	10		
timeout	48		
unused	49		

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Header file for SNIPPETS sorting functions
*/

#ifndef SNIPSORT__H
#define SNIPSORT__H

#include <stddef.h>
#include "dirport.h"

/*
**  Prototypes
*/

#ifdef __STDC__
 #define strsort _strsort
#endif

void hugesort(void HUGE *basep, unsigned  nel,
      unsigned width,
      int (*comp)(void HUGE *, void HUGE *));         /* Hugesort.C     */
void    *sortl(void *list, void *(*getnext)(void *),
            void (*setnext)(void *, void *),
            int (*compare)(void *, void *));          /* Ll_Qsort.C     */
void isort(void *base, size_t nmemb, size_t size,
      int (*comp)(const void *, const void *));       /* Rg_Isort.C     */
void qsort(void *, size_t, size_t,
      int (*)(const void *, const void *));           /* Rg_Qsort.C     */
void swap_chars(char *, char *, size_t);              /* Rg_Qsort.C     */
void quicksort(int v[], unsigned n);                  /* Rgiqsort.C     */
void ssort (void *base, size_t nel, size_t width,
      int (*comp)(const void *, const void *));       /* Rg_Ssort.C     */
void strsort(char **v, unsigned n);                   /* Strsort.C      */

/*
**  File: LL_MSORT.C
*/

typedef struct list_struct {
         struct list_struct *next;
         char *key;
         /* other stuff */
         } list;

list *lsort (list *p);


#endif /* SNIPSORT__H */
```

TEXT STATISTICS

1411 characters
50 lines

LEXICAL STATISTICS

14 comments [std-C]
0 comments [C++]
8 preprocessor instructions
0 constants [character]
1 constants [string]
0 constants [numeric]

SYMBOL TABLE

HUGE		21	23+		
SNIPSORT__H			7	8	
__STDC__		17			
_strsort		18			
base		27	33		
basep		21			
comp		23	28	34	
compare		26			
getnext		24			
h	10				
hugesort		21			
isort		27			
key		43			
list		24	45	47+	
list_struct			41	42	
lsort		47			
n	32	35			
nel		21	33		
next		42			
nmemb		27			
p	47				
qsort		29			
quicksort		32			
setnext		25			
size		27			
size_t		27+	29+	31	33+
sortl		24			
ssort		33			
stddef		10			
strsort		18	35		
swap_chars			31		
v	32	35			
width		22	33		

```
/* +++Date last modified: 05-Jul-1997 */

/* SNIPTREE

   Written by Tom Torfs (2:292/516@fidonet.org)
   I hereby donate this code to the public domain

   Builds custom directory structure for SNIPPETS by reading SNIPPETS.NDX

   Requires POSIX-function mkdir(), rest should be ANSI C.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

int main(void)
{
      FILE *fp;
      char buf[100];
      char curpath[80];
      char oldfile[80],newfile[80];
      int i;
      int skiparea;

      puts("SNIPTREE by Tom Torfs");

      if ((fp=fopen("SNIPPETS.NDX","r"))==NULL)
      {
            puts("ERROR: can't open SNIPPETS.NDX for reading");
            return 1;
      }

      while (1)
      {
            do
            {
                  fgets(buf,100,fp);
                  if (feof(fp)) goto finished;
            }
            while (memcmp(buf,"|=========",10))
                  ;
            fgets(buf,100,fp);
            if (feof(fp)) goto finished;
            if (!memcmp(buf,"|=========",10))
            {
                  fgets(buf,100,fp); /* empty */
                  fgets(buf,100,fp); /* section name */
                  fgets(buf,100,fp); /* empty */
                  fgets(buf,100,fp); /* |======================= */
                  fgets(buf,100,fp); /* |======================= */
                  continue;
            }
            strtok(buf,"\n");
            printf("\nFile area: %s\n",buf+2);
            if (!memicmp(buf+2,"Files deleted",13))
            {
                  skiparea = 1;
                  puts("--> SKIPPING");
            }
            else
            {
                  skiparea = 0;
                  printf("Subdirectory (empty=don't move): ");
                  fgets(curpath,80,stdin);
                  i = strlen(curpath);
                  while (i>0 && (curpath[i-1]=='\n' || curpath[i-1]=='\\'
                                 || curpath[i-1]=='/'))
                        i--;
                  curpath[i] = '\0';
                  if (i>0)
                        mkdir(curpath);
            }
            fgets(buf,100,fp); /* |======================= */
            fgets(buf,100,fp); /* | File  O/S  Description */
            fgets(buf,100,fp); /* | ----  ---  ----------- */
            while (1)
            {
                  fgets(buf,100,fp);
                  if (feof(fp)) goto finished;
                  if (buf[0]=='\n')
                        break;
                  if (skiparea)
                        continue;
                  strtok(buf+2," ");
                  strcpy(oldfile,buf+2);
                  if (curpath[0]=='\0' || !stricmp(oldfile,"SNIPPETS.NDX"))
                        printf("%s (not moved)\n",oldfile);
                  else
                  {
                        sprintf(newfile,"%s\\%s",curpath,oldfile);
                        printf("%s -> %s\n",oldfile,newfile);
                        rename(oldfile,newfile);
                  }
```

```
 101 |             }
 102 |       }
 103 | finished:
 104 |
 105 |       puts("\nFinished.");
 106 |
 107 |       fclose(fp);
 108 |
 109 |       return 0;
 110 | }
```

TEXT STATISTICS

```
  3285 characters
   110 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     6 constants [character]
    18 constants [string]
    36 constants [numeric]
```

SYMBOL TABLE

```
    FILE            25
    MSDOS           17
    NULL            34
    __MSDOS__       17
    buf             26      44      47      49      51      53      54      55      56      57      60
            61      62      80      81      82      85      87      91      92
    curpath         27      71      72      73+     74      76      78      93      97
    defined         17+
    fclose         107
    feof            45      50      86
    fgets           44      49      53      54      55      56      57      71      80      81      82
            85
    finished        45      50      86     103
    fopen           34
    fp              25      34      44      45      49      50      53      54      55      56      57
            80      81      82      85      86     107
    h       13      14      15      20
    i       29      72      73+     74      75      76      77
    main            23
    memcmp          47      51
    memicmp         62
    mkdir           78
    newfile         28      97      98      99
    oldfile         28      92      93      94      97      98      99
    printf          61      70      94      98
    puts            32      36      65     105
    rename          99
    skiparea        30      64      69      89
    sprintf         97
    stdin           71
    stdio           13
    stdlib          14
    strcpy          92
    stricmp         93
    string          15
    strlen          72
    strtok          60      91
    unistd          20
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPTYPE.H - Include file for SNIPPETS data types and commonly used macros
  5 | */
  6 |
  7 | #ifndef SNIPTYPE__H
  8 | #define SNIPTYPE__H
  9 |
 10 | #include <stdlib.h>                          /* For free()           */
 11 | #include <string.h>                          /* For NULL & strlen()  */
 12 |
 13 | typedef enum {Error_ = -1, Success_, False_ = 0, True_} Boolean_T;
 14 |
 15 | #if !defined(WIN32) && !defined(_WIN32) && !defined(__NT__) \
 16 |       && !defined(_WINDOWS)
 17 |  #if !defined(OS2)
 18 |   typedef unsigned char  BYTE;
 19 |   typedef unsigned long  DWORD;
 20 |  #endif
 21 |  typedef unsigned short WORD;
 22 | #else
 23 |  #define WIN32_LEAN_AND_MEAN
 24 |  #define NOGDI
 25 |  #define NOSERVICE
 26 |  #undef INC_OLE1
 27 |  #undef INC_OLE2
 28 |  #include <windows.h>
 29 |  #define HUGE
 30 | #endif
 31 |
 32 | #define NUL '\0'
 33 | #define LAST_CHAR(s) (((char *)s)[strlen(s) - 1])
 34 | #define TOBOOL(x) (!(!(x)))
 35 | #define FREE(p) (free(p),(p)=NULL)
 36 |
 37 | #endif /* SNIPTYPE__H */
```

TEXT STATISTICS

```
   938 characters
    37 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
    21 preprocessor instructions
     1 constants [character]
     0 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    BYTE            18
    Boolean_T       13
    DWORD           19
    Error_          13
    FREE            35
    False_          13
    HUGE            29
    INC_OLE1        26
    INC_OLE2        27
    LAST_CHAR       33
    NOGDI           24
    NOSERVICE       25
    NUL             32
    NULL            35
    OS2             17
    SNIPTYPE__H              7       8
    Success_        13
    TOBOOL          34
    True_           13
    WIN32           15
    WIN32_LEAN_AND_MEAN             23
    WORD            21
    _WIN32          15
    _WINDOWS        16
    __NT__          15
    defined         15+     16      17
    free            35
    h       10      11      28
    p       35+
    s       33+
    stdlib          10
    string          11
    strlen          33
    windows         28
    x       34+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Header file for SNIPPETS string manipulation functions
*/

#ifndef SNIP_STR__H
#define SNIP_STR__H

#include <stddef.h>                     /* For size_t and NULL        */
#include <string.h>                     /* For strncpy() & memmove()  */
#include "sniptype.h"                   /* For LAST_CHAR() & NUL      */
#include "extkword.h"                   /* For FAR                    */

/*
**  Macros to print proper plurals by Bob Stout
*/

#define plural_text(n) &"s"[(1 == (n))]
#define plural_text2(n) &"es"[(1 == (n)) << 1]
#define plural_text3(n) &"y\0ies"[(1 != (n)) << 1]

/*
**  Safe string macros by Keiichi Nakasato
*/

/* strncpy() variants that are guaranteed to append NUL                */

#define strn1cpy(d,s,n) (strncpy(d,s,n),(d)[n]=0,d)
#define strn0cpy(d,s,n) strn1cpy(d,s,(n)-1)

/* like strcpy, except guaranteed to work with overlapping strings     */

#define strMove(d,s) memmove(d,s,strlen(s)+1)


/*
**  Prototypes
**
**  Note: If compiling strictly conforming ANSI/ISO standard C code, the
**        function names are modified to be compliant.
*/

#if defined(__STDC__) && __STDC__
 #define memmem   memMem
 #define strchcat strChcat
 #define strdel   strDel
 #define strdelch strDelch
 #define strdup   strDup
 #define strecpy  strEcpy
 #define stristr  strIstr
 #define strrepl  strRepl
 #define strrev   strRev
 #define strrpbrk strRpbrk
 #define strupr   strUpr
 #define strlwr   strLwr
#endif

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

void *memmem(const void *buf, const void *pattern,    /* Memmem.C       */
      size_t buflen, size_t len);
char *sstrcpy(char *to, char *from);                   /* Sstrcpy.C      */
char *sstrcat(char *to, char *from);                   /* Sstrcpy.C      */
char *sstrdel(char *s, ...);                           /* Sstrdel.C      */
char *stptok(const char *s, char *tok, size_t toklen,
      char *brk);                                      /* Stptok.C       */
char *strchcat(char *string, int ch, size_t buflen);   /* Strchcat.C     */
char *strdel(char *string, size_t first, size_t len);  /* Strdel.C       */
char *strdelch(char *string, const char *lose);        /* Strdelch.C     */
char *strdup(const char *string);                      /* Strdup.C       */
char *strecpy(char *target, const char *src);          /* Strecpy.C/Asm  */
char *stristr(const char *String,                      /* Stristr.C      */
              const char *Pattern);
char *strrepl(char *Str, size_t BufSiz,
      char *OldStr, char *NewStr);                     /* Strrepl.C      */
char *strrev(char *str);                               /* Strrev.C       */
char *strrpbrk(const char *szString,
      const char *szChars);                            /* Strrpbrk.C     */
char *strupr(char *string);                            /* Strupr.C       */
char *strlwr(char *string);                            /* Strupr.C       */
char *translate(char *string);                         /* Translat.C     */
char *xstrcat(char *des, char *src, ...);              /* Xstrcat.C      */
char *rule_line(char * s, unsigned short len,
      short units, char * digits, char filler);        /* Ruleline.C     */
char *rmallws(char *str);                              /* Rmallws.C      */
char *rmlead(char *str);                               /* Rmlead.C       */
char *rmtrail(char *str);                              /* Rmtrail.C      */
char *trim (char *str);                                /* Trim.C         */
void lv1ws(char *str);                                 /* Lv1Ws.C        */

#if defined(MSDOS) || defined(__MSDOS__)
 void FAR *fmemmem(const void FAR *buf,                /* Fmemmem.C      */
       const void FAR *pattern, long buflen, long len);
#endif

#if defined(__cplusplus) && __cplusplus
 }
```

```
 101 | #endif
 102 |
 103 | #endif /*  SNIP_STR__H */


TEXT STATISTICS

  3965 characters
   103 lines

LEXICAL STATISTICS

    37 comments [std-C]
     0 comments [C++]
    33 preprocessor instructions
     0 constants [character]
     6 constants [string]
     8 constants [numeric]

SYMBOL TABLE

    BufSiz          77
    FAR             95+     96
    MSDOS           94
    NewStr          78
    OldStr          78
    Pattern         76
    SNIP_STR__H              7       8
    Str             77
    String          75
    __MSDOS__       94
    __STDC__        44+
    __cplusplus             59+     99+
    brk             69
    buf             63      95
    buflen          64      70      96
    ch              70
    d       29+     30+     34+
    defined         44      59      94+     99
    des             85
    digits          87
    filler          87
    first           71
    fmemmem         95
    from            65      66
    h       10      11
    len             64      71      86      96
    lose            72
    lv1ws           92
    memMem          45
    memmem          45      63
    memmove         34
    n       19+     20+     21+     29+     30+
    pattern         63      96
    plural_text             19
    plural_text2            20
    plural_text3            21
    rmallws         88
    rmlead          89
    rmtrail         90
    rule_line       86
    s       29+     30+     34+     67      68      86
    size_t          64+     68      70      71+     77
    src             74      85
    sstrcat         66
    sstrcpy         65
    sstrdel         67
    stddef          10
    stptok          68
    str             79      88      89      90      91      92
    strChcat        46
    strDel          47
    strDelch        48
    strDup          49
    strEcpy         50
    strIstr         51
    strLwr          56
    strMove         34
    strRepl         52
    strRev          53
    strRpbrk        54
    strUpr          55
    strchcat        46      70
    strdel          47      71
    strdelch        48      72
    strdup          49      73
    strecpy         50      74
    string          11      70      71      72      73      82      83      84
    stristr         51      75
    strlen          34
    strlwr          56      83
    strn0cpy        30
    strn1cpy        29      30
    strncpy         29
    strrepl         52      77
    strrev          53      79
    strrpbrk        54      80
    strupr          55      82
    szChars         81
    szString        80
```

```
target          74
to              65      66
tok             68
toklen          68
translate       84
trim            91
units           87
xstrcat         85
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Header file for SNIPPETS array allocation functions
  5 | */
  6 |
  7 | #ifndef SNPARRAY__H
  8 | #define SNPARRAY__H
  9 |
 10 | /*
 11 | **  AMALLOC.C
 12 | */
 13 |
 14 | void *amalloc( int esiz, void *initval, int dims, ... );
 15 |
 16 | /*
 17 | **  MDALLOC.C
 18 | */
 19 |
 20 | void *mdalloc(int ndim, int width, ...);
 21 | void mdfree(void *tip, int ndim);
 22 |
 23 | #endif /* SNPARRAY__H */
```

TEXT STATISTICS

```
   370 characters
    23 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    SNPARRAY__H              7       8
    amalloc         14
    dims            14
    esiz            14
    initval         14
    mdalloc         20
    mdfree          21
    ndim            20      21
    tip             21
    width           20
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SNPDOSYS.H - Snippets header file for DOS system-level functions
*/

#ifndef SNPDOSYS__H
#define SNPDOSYS__H

#include <stdio.h>
#include "sniptype.h"
#include "extkword.h"


/*
**  Prototypes
*/

int       isBreakOn(void);                              /* Break.C        */
void      setBreak(int OnOff);                          /* Break.C        */
int       drop_time(void);                              /* Droptime.C     */
unsigned  findIslot(void);                              /* Fndislot.C     */
int       format(char drive, char *switches,
            char *vlabel);                              /* Dosfrmat.C     */
int       is_share(char *arg);                          /* Isshare.C      */
long      memavail(void);                               /* Memavail.C     */
int       getNwLoginName (char * namebuf);              /* Nwlinnam.C     */
int       shell_to_DOS(void);                           /* Shel2Dos.C     */
void FAR *addptr (char FAR *p, unsigned long num);      /* Fptr_Add.C     */
void FAR *farnormal(void FAR *ptr);                     /* Fptr_Add.C     */


/*
**  File: JOYSTICK.C
*/

struct joystick {
        Boolean_T switch_0;
        Boolean_T switch_1;
        Boolean_T switch_2;
        Boolean_T switch_3;

        int     pos_Ax;
        int     pos_Ay;
        int     pos_Bx;
        int     pos_By;
};

extern struct joystick JoyStick;

Boolean_T read_joystick(void);


/*
**  File: OS_ID.C
*/

struct i_os_ver
{
  int maj;
  int min;
};

#define DOS     0
#define OS2     1
#define DV      2
#define WINS    3
#define WIN3    4
#define TOT_OS  5

                        /*   76543210  */
#define is_DOS  0x01    /* b'00000001' */
#define is_OS2  0x02    /* b'00000010' */
#define is_DV   0x04    /* b'00000100' */
#define is_WINS 0x08    /* b'00001000' */
#define is_WIN3 0x10    /* b'00010000' */

#ifndef OS_ID_MAIN
  extern int id_os_type;
  extern int id_os;
  extern const char *id_os_name[TOT_OS];
  extern struct i_os_ver id_os_ver[TOT_OS];
#endif

int  get_os(void);                  /* Determine OS                     */
void t_slice(int t_os);             /* Give up a time slice to the OS   */


/*
**  File: PFOPEN.C
*/

#ifdef unix
 #define SEP_CHARS ":"
#else
 #define SEP_CHARS ";"
#endif

FILE *pfopen(const char *name, const char *mode, const char *dirs);

```

```
 101 |
 102 | /*
 103 | **  File: REDIRECT.C
 104 | */
 105 |
 106 | typedef  struct {
 107 |       char  path[FILENAME_MAX];
 108 |       int   which;
 109 |       int   what;
 110 |       int   oldhandle;
 111 |       int   flag;
 112 | } REDIR_STRUCT;
 113 |
 114 | extern REDIR_STRUCT redir_stdin, redir_stdout, redir_stderr;
 115 |
 116 | void  start_redirect    ( REDIR_STRUCT *s );
 117 | void  stop_redirect     ( REDIR_STRUCT *s );
 118 |
 119 |
 120 | #endif /* SNPDOSYS__H */
```

TEXT STATISTICS

```
  2824 characters
   120 lines
```

LEXICAL STATISTICS

```
    27 comments [std-C]
     0 comments [C++]
    24 preprocessor instructions
     0 constants [character]
     4 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       38      39      40      41      51
    DOS             64
    DV              66
    FAR             29+     30+
    FILE            99
    FILENAME_MAX           107
    JoyStick        49
    OS2             65
    OS_ID_MAIN              78
    OnOff           20
    REDIR_STRUCT           112     114     116     117
    SEP_CHARS       94      96
    SNPDOSYS__H             7       8
    TOT_OS          69      81      82
    WIN3            68
    WINS            67
    addptr          29
    arg             25
    dirs            99
    drive           23
    drop_time       21
    farnormal       30
    findIslot       22
    flag           111
    format          23
    getNwLoginName          27
    get_os          85
    h       10
    i_os_ver        58      82
    id_os           80
    id_os_name              81
    id_os_type              79
    id_os_ver       82
    isBreakOn       19
    is_DOS          72
    is_DV           74
    is_OS2          73
    is_WIN3         76
    is_WINS         75
    is_share        25
    joystick        37      49
    maj             60
    memavail        26
    min             61
    mode            99
    name            99
    namebuf         27
    num             29
    oldhandle      110
    p       29
    path           107
    pfopen          99
    pos_Ax          43
    pos_Ay          44
    pos_Bx          45
    pos_By          46
    ptr             30
    read_joystick           51
    redir_stderr           114
    redir_stdin            114
    redir_stdout           114
    s       116     117
```

```
setBreak         20
shell_to_DOS              28
start_redirect           116
stdio            10
stop_redirect            117
switch_0         38
switch_1         39
switch_2         40
switch_3         41
switches         23
t_os             86
t_slice          86
unix             93
vlabel           24
what            109
which           108
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  Header file for SNIPPETS direct disk I/O functions
   5 | */
   6 |
   7 | #ifndef SNPDKSIO__H
   8 | #define SNPDKSIO__H
   9 |
  10 | #include <stddef.h>
  11 | #include <stdio.h>
  12 | #include "extkword.h"
  13 |
  14 | /*
  15 | **  File: ABSDISKC.C and ABSDISK.ASM
  16 | */
  17 |
  18 | int CDECL absdisk(unsigned char  function,
  19 |                   unsigned short drive,               /* 0 = A:, etc.   */
  20 |                   size_t         number_of_sectors,
  21 |                   size_t         starting_sector,
  22 |                   void *         sector_buffer);
  23 |
  24 | int AbsDiskRead(unsigned short drive,
  25 |                 size_t         num_of_sectors,
  26 |                 size_t         sector,
  27 |                 void          *ptr);
  28 |
  29 | int AbsDiskWrite(unsigned short drive,
  30 |                 size_t         num_of_sectors,
  31 |                 size_t         sector,
  32 |                 void *ptr);
  33 |
  34 | /*
  35 | **  File: HUGEREAD.C
  36 | */
  37 |
  38 | long hugeread(int fh, unsigned char FAR *buf, long size);
  39 | long hugewrite(int fh, unsigned char FAR *buf, long size);
  40 | long hugefread(FILE *fp, char FAR *buf, long size);
  41 | long hugefwrite(FILE *fp, char FAR *buf, long size);
  42 |
  43 |
  44 | #endif /* SNPDKSIO__H */
```

TEXT STATISTICS

```
  1158 characters
    44 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    AbsDiskRead             24
    AbsDiskWrite            29
    CDECL           18
    FAR             38      39      40      41
    FILE            40      41
    SNPDKSIO__H              7       8
    absdisk         18
    buf             38      39      40      41
    drive           19      24      29
    fh              38      39
    fp              40      41
    function        18
    h       10      11
    hugefread       40
    hugefwrite              41
    hugeread        38
    hugewrite       39
    num_of_sectors          25      30
    number_of_sectors       20
    ptr             27      32
    sector          26      31
    sector_buffer           22
    size            38      39      40      41
    size_t          20      21      25      26      30      31
    starting_sector         21
    stddef          10
    stdio           11
```

```
/* +++Date last modified: 05-Jul-1997 */

#ifndef SORTKEY_H_
#define SORTKEY_H_

/*************************************************************************
* @(#)sortkey.h
* @(#)      This file contains examples of two sort order tables:
* @(#)            CaseMatch & CaseIgnore
* @(#)      These tables are used for sorting, comparing and case converting
* @(#)      using non-ASCII charactersets
*
*  Each table containes 256 elements. The sort value of a charater is
*  defined as the value in the characters position in the table.
*
*  EXAMPLE:
*     The character , is to be sorted as an ordinary e
*     The , has the ASCII value 130 (82h). On position 130 in the
*     table you will find an e.
*
*     -         |130      |140   <- ASCII decimal value
*     -       €•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••  <- Standard ASCII sort sequence
*            "Cyea{a}ceeeiii[]E  <- Local sort sequence
*
*
*  Please note the C syntax using \ in combination with ' " and \ :
*           \", \', \\
*
*
*=======================================================================
*
*  Danish/Norwegan sort order:
*     special mapping:
*         •      =   y        š     =  Y
*         w      =   v        W     =  V
*         ‘      =   {        ’ Ž   =  [
*         › ”    =   |        • ”   =  \\
*         †      =   }        •     =  ]
*
*     diacritical charaters to ordinary characters:
*         , Š ˆ ‰  = e
*           … ƒ    = a   (note „ = ‘ sorted as { )
*         ¡ • Œ ‹  = i
*         ¢ • “    = o   (note ” = › sorted as } )
*         £ — –    = u   (note • = y)
*
*  Sorting cent and yen characters might sound strange to you, but the
*  Norwegan and danish character set contains tree extra characters:
*
*     ‘ ’  [î:] a combination of a & e
*     › •  [íÁ:] (o slash like í ) a combination of o & e
*     † •  [†:] a sort of double a
*
*  These characters follow A-Z in order ’, •, • and a-z in order ‘, ›, †
*  I swedish the characters „ & ” are used instead of ‘ and ›. Their sort
*  order is A-Z, •, Ž, ™
*
*
*
*=======================================================================
*
* The following examples of other sort orders are picked up from:
*        TDE, the Thomson-Davis Editor
*                Version 2.10
*             November 13, 1992
*                Frank Davis
*
* Frank Davis claimes, that Pierre Jelenc is the creator of these examples.
*
*
* Standard ASCII sort sequence:

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijkl
mnopqrstuvwxyz{|}~•€•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ•žŸ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸
¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

*
*  English/French/Esperanto sort order:  map accents to unaccented
*
*************************     ******
* CaseMatch

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijkl
mnopqrstuvwxyz{|}~••CueaaaaceeeiiiAAEaAooouuyOU›œ•žYaiounN

*
* CaseIgnore

!"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]^_'abcdefghijkl
mnopqrstuvwxyz{|}~••cueaaaaceeeiiiaaeaaooouuyou›œ•žYaiounn

*
*
*  Swedish/Finnish sort order:  map • = y, w = v, and the accents
*
* CaseMatch

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVVXYZ[\]^_'abcdefghijkl
mnopqrstuvvxyz{|}~••CyeaaaaceeeiiiAAEaAooouuyOY›œ•žYaiounN

```

```
 *
 * CaseIgnore

!"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvvxyz[\]^_'abcdefghijkl
mnopqrstuvvxyz{|}~•cyeaaaaceeeiiiaaeaaooouuyoy›œ•žŸaiounn

 *
 *
 *  German sort order:  slide everything down for „, ”, •, á, and the accents
 *  see asterisks for changes ====>                                        *
 *     *           *             *   * *          *  *         *     *    **
 *
 * Standard ASCII sort sequence:

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijkl
mnopqrstuvwxyz{|}~•€•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ•žŸ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸
¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

 *
 * CaseMatch

!"#$%&'()*+,-./0123456789:;<=>?@ACDEFGHIJKLMNOPRSTUVWYZ[\]^_'abcdfghijklmnop
qrsuvwxz{}~•€•‚ƒ„…†D|ideddgiiimmmBAFdAsts{{€QXœ•žŸ dms{rO§¨©ª«¬-®¯°±²³´µ¶·¸¹
º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáy

 *  see asterisks for changes ====>                                        *
 *     *           *             *   * *          *  *         *     *    **
 *
 * CaseIgnore

!"#$%&'()*+,-./0123456789:;<=>?@dfghijklmnopqrsuvwxz{}~•€•^_'abcdfghijklmnop
qrsuvwxz{}~•€•‚ƒ„…†g|ideddgiiimmmedidAsts{{€t|œ•žŸ dms{rr§¨©ª«¬-®¯°±²³´µ¶·¸¹
º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáy

 *  see asterisks for changes ====>                                        *
 *     *           *             *   * *          *  *         *     *    **
 *
 *
 ****************************************************************************
 *@(#)1993-06-10/Erik Bachmann
 ****************************************************************************/

const unsigned char *CaseMatch  =
/*
Standard ASCII sort sequence:
-        |00                |05                |10                |15
-        \x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F
*/      "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F"
/*
-        |16              |20                |25                |30
-        \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F
*/      "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F"

/*
-       |32       |40       |50       |60       |70
-        !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
*/     " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO"
/*
-       |80       |90         |100      |110      |120
-       PQRSTUVWXYZ[\\]^_\'abcdefghijklmnopqrstuvwxyz{|}~•
*/     "PQRSTUVVXYZ[\\]^_\'abcdefghijklmnopqrstuvvxyz{|}~•"
/*
-         |130      |140       |150      |160      |170
-       €•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ •žŸ ¡¢£¤¥¦§¨©ª«¬-®¯
*/     "Cyea{a}ceeeiii[]E{[o|ouuy\\Y|œ\\žŸaiounN¦§¨©ª«¬-®¯"
/*
-           |180      |190      |200      |210      |220
-       °±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
*/     "°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß"
/*
-             |230      |240      |250
-       àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF
*/     "àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF";


/*=====================================================================*/

const unsigned char *CaseIgnore =
/*
Standard ASCII sort sequence:
-        |00                |05                |10                |15
-        \x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F
*/      "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F"
/*
-        |16              |20                |25                |30
-        \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F
*/      "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F"

/*
-       |32       |40       |50       |60       |70
-        !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
*/     " !\"#$%&\'()*+,-./0123456789:;<=>?@abcdefghijklmno"
/*
-       |80       |90         |100      |110      |120
-       PQRSTUVWXYZ[\\]^_\'abcdefghijklmnopqrstuvwxyz{|}~•
*/     "pqrstuvvxyz[\\]^_\'abcdefghijklmnopqrstuvvxyz{|}~•"
/*
-         |130      |140       |150      |160      |170
-       €•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ •žŸ ¡¢£¤¥¦§¨©ª«¬-®¯
*/     "cyea{a}ceeeiii{}e{{o|ouuy|y|œ|žŸaiounn¦§¨©ª«¬-®¯"
```

```
201 | /*
202 | -           |180      |190      |200      |210      |220
203 | -       °±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
204 | */     "°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß"
205 | /*
206 | -            |230      |240      |250
207 | -       àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF
208 | */     "àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF";
209 |
210 | /*====================================================================*/
211 |
212 | const char *CaseUpper  =
213 | /*
214 | Standard ASCII sort sequence:
215 | -        |00                 |05                 |10                 |15
216 | -        \x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F
217 | */      "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F"
218 | /*
219 | -        |16             |20                 |25                 |30
220 | -        \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F
221 | */      "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F"
222 |
223 | /*
224 | -       |32       |40       |50       |60       |70
225 | -        !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
226 | */     " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO"
227 | /*
228 | -       |80       |90         |100      |110      |120
229 | -       PQRSTUVWXYZ[\\]^_\'abcdefghijklmnopqrstuvwxyz{|}~•
230 | */     "PQRSTUVWXYZ[\\]^_\'ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~•"
231 | /*
232 | -         |130      |140      |150    |160      |170
233 | -       €•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ•žŸ ¡¢£¤¥¦§¨©ª«¬-®¯
234 | */     "€š•ŠŽ·•€ÒÓÔØ×ÞŽ••’’’â™ãêëY™š›œ•žŸµÖàé¥¥¦§¨©ª«¬-®¯"
235 | /*
236 | -           |180      |190      |200      |210      |220
237 | -       °±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
238 | */     "°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÇÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß"
239 | /*
240 | -            |230      |240      |250
241 | -       àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF
242 | */     "àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF";
243 |
244 |
245 | /*====================================================================*/
246 |
247 |
248 | const char *CaseLower  =
249 | /*
250 | Standard ASCII sort sequence:
251 | -        |00                 |05                 |10                 |15
252 | -        \x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F
253 | */      "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F"
254 | /*
255 | -        |16             |20                 |25                 |30
256 | -        \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F
257 | */      "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F"
258 |
259 | /*
260 | -       |32       |40       |50       |60       |70
261 | -        !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
262 | */     " !\"#$%&\'()*+,-./0123456789:;<=>?@abcdefghijklmno"
263 | /*
264 | -       |80       |90         |100      |110      |120
265 | -       PQRSTUVWXYZ[\\]^_\'abcdefghijklmnopqrstuvwxyz{|}~•
266 | */     "pqrstuvwxyz[\\]^_\'abcdefghijklmnopqrstuvwxyz{|}~•"
267 | /*
268 | -         |130      |140      |150    |160      |170
269 | -       €•‚ƒ„…†‡ˆ‰Š‹Œ•Ž••‘’“”•–—˜™š›œ•žŸ ¡¢£¤¥¦§¨©ª«¬-®¯
270 | */     "‡•‚ƒ„…†‡ˆ‰Š‹Œ•„†‚‘’“”•–—˜”•›œ›žŸ ¡¢£¤¦§¨©ª«¬-®¯"
271 | /*
272 | -           |180      |190      |200      |210      |220
273 | -       °±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
274 | */     "°±²³´ ƒ…¸¹º»¼½¾¿ÀÁÂÃÄÅÆÆÈÉÊËÌÍÎÏÐÑˆ‰ŠÕ¡Œ‹ÙÚÛÜÝ•ß"
275 | /*
276 | -            |230      |240      |250
277 | -       àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ\xFF
278 | */     "¢á“•äåæèè£——ìíîïðñòóôõö÷øùúûüýþ\xFF";
279 |
280 | #endif /*  SORTKEY_H_ */
```

```
TEXT STATISTICS

 10705 characters
   280 lines

LEXICAL STATISTICS

    34 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
    28 constants [string]
     0 constants [numeric]

SYMBOL TABLE

    CaseIgnore             178
    CaseLower      248
    CaseMatch      143
    CaseUpper      212
    SORTKEY_H_               3       4
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SOUND.C
  5 | **
  6 | **  Original Copyright 1988-1991 by Bob Stout as part of
  7 | **  the MicroFirm Function Library (MFL)
  8 | **
  9 | **  The user is granted a free limited license to use this source file
 10 | **  to create royalty-free programs, subject to the terms of the
 11 | **  license restrictions specified in the LICENSE.MFL file.
 12 | */
 13 |
 14 | #include "pchwio.h"
 15 | #include "sound.h"
 16 |
 17 | void soundon(void)
 18 | {
 19 |       short value;
 20 |
 21 |       value = inp(SCNTRL);
 22 |       value |= SOUNDON;
 23 |       outp(SCNTRL, value);
 24 | }
 25 |
 26 | void soundoff(void)
 27 | {
 28 |       short value;
 29 |
 30 |       value = inp(SCNTRL);
 31 |       value &= SOUNDOFF;
 32 |       outp(SCNTRL, value);
 33 | }
```

TEXT STATISTICS

```
   679 characters
    33 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

SCNTRL	21	23	30	32			
SOUNDOFF	31						
SOUNDON	22						
inp	21	30					
outp	23	32					
soundoff	26						
soundon	17						
value	19	21	22	23	28	30	31	32

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SOUND.H
**
**  Original Copyright 1988-1991 by Bob Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
*/

#ifndef SOUND__H
#define SOUND__H

#if defined(__ZTC__)
 #include <int.h>
 #undef int_on
 #undef int_off
#elif defined(__TURBOC__)
 #define int_on         enable
 #define int_off        disable
 #ifndef inp
  #define inp           inportb
 #endif
 #ifndef outp
  #define outp          outportb
 #endif
#else /* assume MSC/QC */
 #include <conio.h>
 #define int_on         _enable
 #define int_off        _disable
 #define getvect        _dos_getvect
 #define setvect        _dos_setvect
#endif

/* defines for mktone() update parameter: */

#define ON 0          /* turn the speaker on */
#define UPDATE 1      /* sound is on, just change freq */
#define TOGGLE 2      /* for delay use, turn on, then off */

/* port equates */

#define SCNTRL 0x61   /* sound control port */
#define SOUNDON 0x03  /* bit mask to enable speaker */
#define SOUNDOFF 0xfc /* bit mask to disable speaker */
#define C8253 0x43    /* port address to control 8253 */
#define SETIMER 0xb6  /* tell 8253 to expect freq data next */
#define F8253 0x42    /* frequency address on 8253 */

/* frequency equates (musical scale) */
/* digit in label is octave number, S indicates Sharp (#) */

#define C0      36489
#define CS0     34445
#define D0      32512
#define DS0     30673
#define E0      28961
#define F0      27329
#define FS0     25804
#define G0      24351
#define GS0     22981
#define A0      21694
#define AS0     20473
#define B0      19326

#define C1      18244
#define CS1     17218
#define D1      16251
#define DS1     15340
#define E1      14480
#define F1      13668
#define FS1     12899
#define G1      12175
#define GS1     11493
#define A1      10847
#define AS1     10238
#define B1      9663

#define C2      9121
#define CS2     8609
#define D2      8126
#define DS2     7670
#define E2      7239
#define F2      6833
#define FS2     6450
#define G2      6088
#define GS2     5746
#define A2      5424
#define AS2     5119
#define B2      4832

#define C3      4561
#define CS3     4305
#define D3      4063
#define DS3     3835
#define E3      3620
#define F3      3417
```

```
101 | #define FS3     3225
102 | #define G3      3044
103 | #define GS3     2873
104 | #define A3      2712
105 | #define AS3     2560
106 | #define B3      2416
107 |
108 | #define C4      2280
109 | #define CS4     2152
110 | #define D4      2032
111 | #define DS4     1917
112 | #define E4      1810
113 | #define F4      1708
114 | #define FS4     1612
115 | #define G4      1522
116 | #define GS4     1437
117 | #define A4      1356
118 | #define AS4     1280
119 | #define B4      1210
120 |
121 | #define C5      1140
122 | #define CS5     1076
123 | #define D5      1016
124 | #define DS5     959
125 | #define E5      905
126 | #define F5      854
127 | #define FS5     806
128 | #define G5      761
129 | #define GS5     718
130 | #define A5      678
131 | #define AS5     640
132 | #define B5      604
133 |
134 | #define C6      570
135 | #define CS6     538
136 | #define D6      508
137 | #define DS6     479
138 | #define E6      449
139 | #define F6      427
140 | #define FS6     403
141 | #define G6      380
142 | #define GS6     359
143 | #define A6      339
144 | #define AS6     320
145 | #define B6      302
146 |
147 | #define C7      285
148 | #define CS7     269
149 | #define D7      254
150 | #define DS7     240
151 | #define E7      226
152 | #define F7      214
153 | #define FS7     202
154 | #define G7      190
155 | #define GS7     180
156 | #define A7      169
157 | #define AS7     160
158 | #define B7      151
159 |
160 | #define C8      143
161 |
162 | #define REST    0
163 |
164 | typedef struct
165 | {
166 |         unsigned int      freq;
167 |         unsigned int      duration;
168 | } NOTE;
169 |
170 | #if defined(__cplusplus) && __cplusplus
171 |  extern "C" {
172 | #endif
173 |
174 | void  mktone(int, int, unsigned),
175 |       dosound(int),
176 |       soundon(void),
177 |       soundoff(void),
178 |       playb_close(void);
179 |
180 | int   playb_note(unsigned, unsigned);
181 |
182 | NOTE *playb_open(unsigned);
183 |
184 | #if defined(__cplusplus) && __cplusplus
185 |  }
186 | #endif
187 |
188 | #endif /* SOUND__H */
```

TEXT STATISTICS

4187 characters
188 lines

LEXICAL STATISTICS

17 comments [std-C]
0 comments [C++]
134 preprocessor instructions
0 constants [character]
1 constants [string]
107 constants [numeric]

SYMBOL TABLE

A0	65
A1	78
A2	91
A3	104
A4	117
A5	130
A6	143
A7	156
AS0	66
AS1	79
AS2	92
AS3	105
AS4	118
AS5	131
AS6	144
AS7	157
B0	67
B1	80
B2	93
B3	106
B4	119
B5	132
B6	145
B7	158
C0	56
C1	69
C2	82
C3	95
C4	108
C5	121
C6	134
C7	147
C8	160
C8253	49
CS0	57
CS1	70
CS2	83
CS3	96
CS4	109
CS5	122
CS6	135
CS7	148
D0	58
D1	71
D2	84
D3	97
D4	110
D5	123
D6	136
D7	149
DS0	59
DS1	72
DS2	85
DS3	98
DS4	111
DS5	124
DS6	137
DS7	150
E0	60
E1	73
E2	86
E3	99
E4	112
E5	125
E6	138
E7	151
F0	61
F1	74
F2	87
F3	100
F4	113
F5	126
F6	139
F7	152
F8253	51
FS0	62
FS1	75
FS2	88
FS3	101
FS4	114
FS5	127
FS6	140
FS7	153
G0	63

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Public domain from Bob Jarvis
*/

#include <stdio.h>
#include <ctype.h>
#include "phonetic.h"

char *soundex(char *instr, char *outstr)
{                   /* ABCDEFGHIJKLMNOPQRSTUVWXYZ */
        char *table = "01230120022455012623010202";
        char *outptr = outstr;
        int count = 0;

        while(!isalpha(instr[0]) && instr[0])
                ++instr;

        if(!instr[0])     /* Hey!  Where'd the string go? */
                return(NULL);

        if(toupper(instr[0]) == 'P' && toupper(instr[1]) == 'H')
        {
                instr[0] = 'F';
                instr[1] = 'A';
        }

        *outptr++ = (char)toupper(*instr++);

        while(*instr && count < 5)
        {
                if(isalpha(*instr) && *instr != *(instr-1))
                {
                        *outptr = table[toupper(instr[0]) - 'A'];
                        if(*outptr != '0')
                        {
                                ++outptr;
                                ++count;
                        }
                }
                ++instr;
        }

        *outptr = '\0';
        return(outstr);
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      char code[6];

      if (argc != 2)
      {
            puts("Usage: SOUNDEX string");
            return EXIT_FAILURE;
      }

      printf("soundex(\"%s\") returned %s\n",
            argv[1], soundex(argv[1], code));

      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1606 characters
    70 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     7 constants [character]
     4 constants [string]
    15 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            61
    EXIT_SUCCESS            67
    NULL            21
    TEST            49
    argc            54      58
    argv            54      65+
    code            56      65
    count           15      31      39
    ctype            8
    h        7       8      51      52
    instr           11      17+     18      20      23+     25      26      29      31      33+     35
            42
    isalpha         17      33
    main            54
    outptr          14      29      35      36      38      45
    outstr          11      14      46
    printf          64
    puts            60
    soundex         11      65
    stdio            7      51
    stdlib          52
    table           13      35
    toupper         23+     29      35
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Variant Soundex Algorithm (Algorithm #4), Optimized for use with Surnames
**
**  Written by Joe Celko. Originally published in "The C Gazette" vol. 4 nr. 2
**  Source code may be freely used is source is acknowledged
**  Object code may be freely used
**
**  Uses STRUPR.C, also from SNIPPETS
*/

/*
**  Preserves first character, uppercase name, drop non-alpha characters,
**  convert letters to Soundex digits, and returns first N letters.
**  Many of the steps here could be combined into the same loop, but
**  they are kept separate for clarity and to give the user a chance
**  to experiment with changes.
*/

#include <string.h>
#include "phonetic.h"
#include "snip_str.h"

#define WBUFSIZE 512

void soundex4(const char *instr, char *outstr, int N)
{
      char *p, *p1;
      int i;
      char workbuf[WBUFSIZE + 1];
      char priorletter;

      /* Make a working copy  */

      strncpy(workbuf, instr, WBUFSIZE);
      workbuf[WBUFSIZE] = NUL;
      strupr(workbuf);

      /* Convert all vowels to 'A'  */

      for (p = workbuf; *p; ++p)
      {
            if (strchr("AEIOUY", *p))
                  *p = 'A';
      }

      /* Prefix transformations: done only once on the front of a name  */

      if (Success_ == strncmp(workbuf, "MAC", 3))     /* MAC to MCC     */
            workbuf[1] = 'C';
      else if (Success_ == strncmp(workbuf, "KN", 2)) /* KN to NN       */
            workbuf[0] = 'N';
      else if ('K' == workbuf[0])                     /* K to C         */
            workbuf[0] = 'C';
      else if (Success_ == strncmp(workbuf, "PF", 2)) /* PF to FF       */
            workbuf[0] = 'F';
      else if (Success_ == strncmp(workbuf, "SCH", 3))/* SCH to SSS     */
            workbuf[1] = workbuf[2] = 'S';

      /*
      ** Infix transformations: done after the first letter,
      ** left to right
      */

      while ((p = strstr(workbuf, "DG")) > workbuf)   /* DG to GG       */
            p[0] = 'G';
      while ((p = strstr(workbuf, "CAAN")) > workbuf) /* CAAN to TAAN   */
            p[0] = 'T';
      while ((p = strchr(workbuf, 'D')) > workbuf)    /* D to T         */
            p[0] = 'T';
      while ((p = strstr(workbuf, "NST")) > workbuf)  /* NST to NSS     */
            p[2] = 'S';
      while ((p = strstr(workbuf, "AV")) > workbuf)   /* AV to AF       */
            p[1] = 'F';
      while ((p = strchr(workbuf, 'Q')) > workbuf)    /* Q to G         */
            p[0] = 'G';
      while ((p = strchr(workbuf, 'Z')) > workbuf)    /* Z to S         */
            p[0] = 'S';
      while ((p = strchr(workbuf, 'M')) > workbuf)    /* M to N         */
            p[0] = 'N';
      while ((p = strstr(workbuf, "KN")) > workbuf)   /* KN to NN       */
            p[0] = 'N';
      while ((p = strchr(workbuf, 'K')) > workbuf)    /* K to C         */
            p[0] = 'C';
      while ((p = strstr(workbuf, "AH")) > workbuf)   /* AH to AA       */
            p[1] = 'A';
      while ((p = strstr(workbuf, "HA")) > workbuf)   /* HA to AA       */
            p[0] = 'A';
      while ((p = strstr(workbuf, "AW")) > workbuf)   /* AW to AA       */
            p[1] = 'A';
      while ((p = strstr(workbuf, "PH")) > workbuf)   /* PH to FF       */
            p[0] = p[1] = 'F';
      while ((p = strstr(workbuf, "SCH")) > workbuf)  /* SCH to SSS     */
            p[0] = p[1] = 'S';

      /*
      ** Suffix transformations: done on the end of the word,
      ** right to left
      */
```

```
101 |
102 |       /* (1) remove terminal 'A's and 'S's      */
103 |
104 |       for (i = strlen(workbuf) - 1;
105 |             (i > 0) && ('A' == workbuf[i] || 'S' == workbuf[i]);
106 |             --i)
107 |       {
108 |             workbuf[i] = NUL;
109 |       }
110 |
111 |       /* (2) terminal NT to TT      */
112 |
113 |       for (i = strlen(workbuf) - 1;
114 |             (i > 1) && ('N' == workbuf[i - 1] || 'T' == workbuf[i]);
115 |             --i)
116 |       {
117 |             workbuf[i - 1] = 'T';
118 |       }
119 |
120 |       /* Now strip out all the vowels except the first     */
121 |
122 |       p = p1 = workbuf;
123 |       while (NUL != (*p1++ = *p++))
124 |       {
125 |             while ('A' == *p)
126 |                   ++p;
127 |       }
128 |
129 |       /* Remove all duplicate letters     */
130 |
131 |       p = p1 = workbuf;
132 |       priorletter = NUL;
133 |       do {
134 |             while (*p == priorletter)
135 |                   ++p;
136 |             priorletter = *p;
137 |       } while (NUL != (*p1++ = *p++));
138 |
139 |       /* Finish up      */
140 |
141 |       strncpy(outstr, workbuf, N);
142 |       outstr[N] = NUL;
143 | }
144 |
145 | #ifdef TEST
146 |
147 | #include <stdio.h>
148 | #include <stdlib.h>
149 |
150 | main(int argc, char *argv[])
151 | {
152 |       char outbuf[80];
153 |       int N;
154 |
155 |       if (argc != 3)
156 |       {
157 |             puts("Usage: SOUNDEX4 string length");
158 |             return EXIT_FAILURE;
159 |       }
160 |
161 |       soundex4(argv[1], outbuf, N = atoi(argv[2]));
162 |
163 |       printf("soundex4(\"%s\", %d) => \"%s\"\n", argv[1], N, outbuf);
164 |
165 |       return EXIT_SUCCESS;
166 | }
167 |
168 | #endif /* TEST */
```

```
TEXT STATISTICS

  5040 characters
   168 lines

LEXICAL STATISTICS

    34 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
    33 constants [character]
    19 constants [string]
    41 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE          158
    EXIT_SUCCESS          165
    N       27    141     142     153     161     163
    NUL            37     108     123     132     137     142
    Success_       50      52      56      58
    TEST          145
    WBUFSIZE       25      31      36      37
    argc          150     155
    argv          150    161+     163
    atoi          161
    h       21    147     148
    i       30    104    105+     106     108     113    114+     115     117
    instr          27      36
    main          150
    outbuf        152     161     163
    outstr         27     141     142
    p       29     42+     44      45      66      67      68      69      70      71      72      73
            74      75      76      77      78      79      80      81      82      83      84      85
            86      87      88      89      90      91      92     93+      94     95+     122     123
           125     126     131     134     135     136     137
    p1             29     122     123     131     137
    printf        163
    priorletter            32     132     134     136
    puts          157
    soundex4       27     161
    stdio         147
    stdlib        148
    strchr         44      70      76      78      80      84
    string         21
    strlen        104     113
    strncmp        50      52      56      58
    strncpy        36     141
    strstr         66      68      72      74      82      86      88      90      92      94
    strupr         38
    workbuf        31      36      37      38      42      50      51      52      53      54      55
            56     57      58     59+     66+     68+     70+     72+     74+     76+     78+     80+
           82+    84+     86+     88+     90+     92+     94+     104     105+    108     113     114+
          117     122     131     141
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  S O U N D E X 5
**
**  Originally from SNIPPETS, written by Bob Jarvis.
**  Modified by M. Stapleton of Graphic Bits on Dec  8 1994
**  Uses compressed heptal - eliminates codes for non-trailing zeros.
*/

#include <stdlib.h>
#include <ctype.h>
#include "phonetic.h"

int soundex5(char *instr)
{
      int count = 0, init, snd = 0, notlast = 1;
      int ch;
      static int table[] =
      {
      /* A  B  C  D  E     F  G  H  I  J */
         0, 1, 2, 3, 0,    1, 2, 0, 0, 2,

      /* K  L  M  N  O     P  Q  R  S  T */
         2, 4, 5, 5, 0,    1, 2, 6, 2, 3,

      /* U  V  W  X  Y  Z */
         0, 1, 0, 2, 0, 2,
      };

      /* Skip leading non-alpha */

      while(*instr && (!isalpha(*instr)))
            instr++;

      if(!*instr)     /* Hey!  Where'd the string go? */
            return -1;

      /* Convert initial letter to int. */

      init = (char)toupper(instr[0]) - 'A';

      for(instr++; *instr && (count < 3); instr++)
      {
            if(isalpha(*instr) && (*instr != *(instr-1)))
            {
                  if(NUL != (ch = table[toupper(*instr) - 'A']))
                  {
                        /* Convert to "compressed heptal" */

                        snd = (7 - notlast) * snd + ch - notlast;
                        notlast = ++count < 2;
                  }
            }
      }

      /* Adjust */

      switch(count)
      {
      case 0:     /* default:    Shouldn't get here! */
            snd = 0;
            break;

      case 1:
            snd += 1;
            break;

      case 2:
            snd *= 7;
            /* Fall through */

      case 3:
            snd += 7;
            break;
      }
      return SNDMAX * init + snd;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      if (argc != 2)
      {
            puts("Usage: SOUNDEX5 string");
            return EXIT_FAILURE;
      }

      printf("soundex5(\"%s\") returned %d\n", argv[1], soundex5(argv[1]));

      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2178 characters
    98 lines

LEXICAL STATISTICS

    13 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     2 constants [character]
     3 constants [string]
    46 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            90
    EXIT_SUCCESS            95
    NUL             47
    SNDMAX          77
    TEST            80
    argc            85      87
    argv            85      93+
    ch              18      47      51
    count           17      43      52      59
    ctype           12
    h       11      12      82      83
    init            17      41      77
    instr           15      33+     34      36      41      43+     45+     47
    isalpha         33      45
    main            85
    notlast         17      51+     52
    printf          93
    puts            89
    snd             17      51+     62      66      70      74      77
    soundex5        15      93
    stdio           82
    stdlib          11      83
    table           19      47
    toupper         41      47
```

```
/* +++Date last modified: 05-Jul-1997 */

/******** CALCULATE I/O PERFORMANCE TO NUL FILE ********/

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "errors.h"           /* For cant()     */

#define CHK 100L              /* speed factor   */

long ticks(long tick)                                   /* GET BIOS TIME TICK */
{
        union REGS reg;
        reg.h.ah=0;
        int86(0x1A, &reg, &reg);
        return ((long)reg.x.cx<<16)+reg.x.dx-tick;
}

long time_it(void(*func)(void))
{
        long t = ticks(0L);
        (*func)();
        return ticks(t);
}

void show_it(long t)
{
        long lquot, lrem;
        t = (t*1000/182+5)/10;
        lquot = t/10;
        lrem = t%10;
        printf("%3ld.%02d sec", lquot, (int)lrem);
}

void t_printf(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wt");
        for (u=0; u<50*CHK; ++u)
                fprintf(fp, "Now is %d time for %d little indians\n", 123, -456);
        fclose(fp);
}

void b_printf(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wb");
        for (u=0; u<50*CHK; ++u)
                fprintf(fp, "Now is %d time for %d little indians\n", 123, -456);
        fclose(fp);
}

void tu_printf(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wt");
        setbuf(fp, NULL);
        for (u=0; u<5*CHK; ++u)
                fprintf(fp, "Now is %d time for %d little indians\n", 123, -456);
        fclose(fp);
}

void bu_printf(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wb");
        setbuf(fp, NULL);
        for (u=0; u<5*CHK; ++u)
                fprintf(fp, "Now is %d time for %d little indians\n", 123, -456);
        fclose(fp);
}

void t_write(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wt");
        for (u=0; u<250*CHK; ++u)
                fwrite("Now is the time for all good men to come\n", 41, 1, fp);
        fclose(fp);
}

void b_write(void)
{
        register FILE *fp;
        register unsigned u;

        fp = cant("NUL", "wb");
        for (u=0; u<500*CHK; ++u)
                fwrite("Now is the time for all good men to come\n", 41, 1, fp);
```

```
101 |         fclose(fp);
102 | }
103 |
104 | void tu_write(void)
105 | {
106 |         register FILE *fp;
107 |         register unsigned u;
108 |
109 |         fp = cant("NUL", "wt");
110 |         setbuf(fp, NULL);
111 |         for (u=0; u<100*CHK; ++u)
112 |                 fwrite("Now is the time for all good men to come\n", 41, 1, fp);
113 |         fclose(fp);
114 | }
115 |
116 | void bu_write(void)
117 | {
118 |         register FILE *fp;
119 |         register unsigned u;
120 |
121 |         fp = cant("NUL", "wb");
122 |         setbuf(fp, NULL);
123 |         for (u=0; u<200*CHK; ++u)
124 |                 fwrite("Now is the time for all good men to come\n", 41, 1, fp);
125 |         fclose(fp);
126 | }
127 |
128 | main(void)
129 | {
130 |         show_it(time_it(t_printf));
131 |         printf(": time for text printf buffered\n");
132 |         show_it(time_it(b_printf));
133 |         printf(": time for binary printf buffered\n");
134 |         show_it(time_it(tu_printf));
135 |         printf(": time for text printf unbuffered\n");
136 |         show_it(time_it(bu_printf));
137 |         printf(": time for binary printf unbuffered\n");
138 |
139 |         show_it(time_it(t_write));
140 |         printf(": time for text write buffered\n");
141 |         show_it(time_it(b_write));
142 |         printf(": time for binary write buffered\n");
143 |         show_it(time_it(tu_write));
144 |         printf(": time for text write unbuffered\n");
145 |         show_it(time_it(bu_write));
146 |         printf(": time for binary write unbuffered\n");
147 |
148 |         return 0;
149 | }
```

TEXT STATISTICS

```
  3763 characters
   149 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
    34 constants [string]
    43 constants [numeric]
```

SYMBOL TABLE

```
    CHK             10      42      53      65      77      88      99     111     123
    FILE            38      49      60      72      84      95     106     118
    NULL            64      76     110     122
    REGS            14
    ah              15
    b_printf        47     132
    b_write         93     141
    bu_printf       70     136
    bu_write       116     145
    cant            41      52      63      75      87      98     109     121
    cx              17
    dos              5
    dx              17
    fclose          44      55      67      79      90     101     113     125
    fp              38      41      43      44      49      52      54      55      60      63      64
           66       67      72      75      76      78      79      84      87      89      90      95
           98      100     101     106     109     110     112     113     118     121     122     124
          125
    fprintf         43      54      66      78
    func            20      23
    fwrite          89     100     112     124
    h        5       6       7      15
    int86           16
    lquot           29      31      33
    lrem            29      32      33
    main           128
    printf          33     131     133     135     137     140     142     144     146
    reg             14      15     16+     17+
    setbuf          64      76     110     122
    show_it         27     130     132     134     136     139     141     143     145
    stdio            6
    stdlib           7
    t       22      24      27     30+      31      32
    t_printf        36     130
    t_write         82     139
    tick            12      17
    ticks           12      22      24
    time_it         20     130     132     134     136     139     141     143     145
    tu_printf       58     134
    tu_write       104     143
    u       39      42+     50      53+     61      65+     73      77+     85      88+     96      99+
           107     111+    119     123+
    x       17+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 This program was based on a Pascal program posted in the FIDO 80XXX
 assembly conference.  That Pascal program had the following comment:

                     ------------------------
 This program, which produces the first 1000 digits of PI, using
 only integer arithmetic, comes from an article in the "American
 Mathematical Monthly", Volume 102, Number 3, March 1995, page
 195, by Stanley Rabinowitz and Stan Wagon.
                      ------------------------

 My C implementation is placed into the public domain, by its author
 Carey Bloodworth, on August 22, 1996.

 I have not seen the original article, only that Pascal implementation,
 but based on a discussion in the 80XXX conference, I believe the
 program should be accurate to at least 32 million digits when using
 long unsigned integers.  Using only 16 bit integers causes the
 variables to overflow after a few hundred digits.
*/

#include <stdio.h>

long unsigned int i, j, k, nines, predigit;
long unsigned int q, x, numdig, len;
long unsigned int *pi;

int printed = 0, line = 1;

void OutDig(int dig)
{
      putchar(dig);
      printed++;
      if ((printed%50) == 0)
      {
            printed = 0;
            printf("\nL%04d:   ", line++);
      }
      else if ((printed%10) == 0)putchar(' ');
}

int main(int argc, char *argv[])
{
      if (argc < 2)
      {
            printf("I need to know how many digits to compute\n");
            return 1;
      }

      numdig = atol(argv[1]);
      len = (numdig*10)/3;

      pi = (long unsigned int *)malloc((len+1)*sizeof(long unsigned int) );
      if (pi == NULL)
      {
            printf("Unable to allocate memory\n");
            return 1;
      }

      for (x = len; x > 0; x--)
            pi[x] = 2;

      printf("L0001: 3.");

      nines = 0;
      predigit = 0;
      for (j = 0; j <= numdig; j++)
      {
            q = 0;
            for (i = len; i > 0; i--)
            {
                  x = 10 * pi[i]+ q * i;
                  pi[i] = x % (2 * i - 1);
                  q = x / (2 * i - 1);
            }
            pi[1] = q % 10; q = q / 10;
            if (q == 9)
                  nines++;
            else if (q == 10)
            {
                  OutDig('1' + predigit);
                  while (nines)
                  {
                        OutDig('0');
                        nines--;
                  }
                  predigit = 0;
                  fflush(stdout);
            }
            else
            {
                  if (j > 1)
                        OutDig('0' + predigit);
                  while (nines)
                  {
                        OutDig('9');
                        nines--;
                  }
```

```
 101 |                   predigit = q;
 102 |                   fflush(stdout);
 103 |               }
 104 |       }
 105 |       OutDig(predigit + '0');
 106 |
 107 |       free(pi);
 108 |       return 0;
 109 | }
```

TEXT STATISTICS

```
  3015 characters
   109 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     6 constants [character]
     4 constants [string]
    34 constants [numeric]
```

SYMBOL TABLE

```
    NULL            56
    OutDig          32      83      86      95      98     105
    argc            44      46
    argv            44      52
    atol            52
    dig             32      34
    fflush          90     102
    free           107
    h       24
    i       26      72+     74+     75+     76
    j       26      69+     94
    k       26
    len             27      53      55      62      72
    line            30      39
    main            44
    malloc          55
    nines           26      67      80      84      87      96      99
    numdig          27      52      53      69
    pi              28      55      56      63      74      75      78     107
    predigit        26      68      83      89      95     101     105
    printed         30      35      36      38      41
    printf          39      48      58      65
    putchar         34      41
    q       27      71      74      76      78+     79      81     101
    stdio           24
    stdout          90     102
    x       27      62+     63      74      75      76
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  Miscellaneous text spinners demonstration
 5 | **
 6 | **  public domain by Jon Guthrie, Bob Stout, and others
 7 | */
 8 |
 9 | #include <stdio.h>
10 |
11 | #define DURATION 500    /* Length of demo       */
12 | #define SSLOWDOWN 15    /* Make spinner look ok */
13 | #define TSLOWDOWN 20    /* Make target look ok  */
14 |
15 | main()
16 | {
17 |       unsigned i;
18 |       char spinner[] = "|/-\\", target[] = ".oO";
19 |
20 |       for (i = 0; i < DURATION; ++i)
21 |       {
22 |             unsigned scount = i / SSLOWDOWN, tcount = i / TSLOWDOWN;
23 |             unsigned scountdown  = DURATION / SSLOWDOWN;
24 |             unsigned tcountdown  = DURATION / TSLOWDOWN;
25 |
26 |             printf("CW %c ... CCW %c ... Explode %c ... Implode %c\r",
27 |                   spinner[scount & 3], spinner[(scountdown - scount) & 3],
28 |                   target[tcount % 3], target[(tcountdown - tcount) % 3]);
29 |       }
30 |       return 0;
31 | }
```

TEXT STATISTICS

```
   912 characters
    31 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
     9 constants [numeric]
```

SYMBOL TABLE

```
    DURATION        11      20      23      24
    SSLOWDOWN       12      22      23
    TSLOWDOWN       13      22      24
    h        9
    i       17      20+     22+
    main            15
    printf          26
    scount          22      27+
    scountdown              23      27
    spinner         18      27+
    stdio            9
    target          18      28+
    tcount          22      28+
    tcountdown              24      28
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SPLIT.C - A utility to split large text files into smaller files
**
**  public domain by Bob Stout
**
**  uses FNSPLIT.C from SNIPPETS
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "extkword.h"
#include "filnames.h"

int main(int argc, char *argv[])
{
      long newsize = 32L * 1024L;
      size_t seq = 0;
      char fname[FILENAME_MAX];
      FILE *from;

      if (2 > argc)
      {
            puts("SPLIT big_file [size_in_K]\n");
            puts("creates files of the same name, "
                  "but with numeric extensions");
            puts("a maximum file size may be specified for new files");
            return EXIT_SUCCESS;
      }
      if (2 < argc)
      {
            newsize   = atol(argv[2]);
            newsize <<= 10;
      }
      if (NULL == (from = fopen(argv[1], "r")))
      {
            printf("\aSPLIT: error - can't open %s\n", argv[1]);
            return EXIT_FAILURE;
      }
      fnSplit(argv[1], NULL, NULL, NULL, NULL, fname, NULL);
      while (!feof(from))
      {
            char newname[FILENAME_MAX], buf[1024];
            FILE *to;
            long bytes;

            sprintf(newname, "%s.%03d", fname, seq++);
            if (NULL == (to = fopen(newname, "w")))
            {
                  printf("\aSPLIT: error - can't write %s\n", newname);
                  return EXIT_FAILURE;
            }
            for (bytes = 0L; !feof(from) && (bytes < newsize); )
            {
                  if (fgets(buf, 1023, from))
                  {
                        fputs(buf, to);
                        bytes += (long)strlen(buf);
                  }
            }
            fclose(to);
            printf("%s written\n", newname);
      }
      return EXIT_SUCCESS;
}
```

TEXT STATISTICS

```
  1875 characters
    67 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
    12 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            40      53
    EXIT_SUCCESS            30      66
    FILE            22      46
    FILENAME_MAX            21      45
    NULL            37      42+     50
    argc            17      24      32
    argv            17      34      37      39      42
    atol            34
    buf             45      57      59      60
    bytes           47      55+     60
    fclose          63
    feof            43      55
    fgets           57
    fnSplit         42
    fname           21      42      49
    fopen           37      50
    fputs           59
    from            22      37      43      55      57
    h       11      12      13
    main            17
    newname         45      49      50      52      64
    newsize         19      34      35      55
    printf          39      52      64
    puts            26      27      29
    seq             20      49
    size_t          20
    sprintf         49
    stdio           11
    stdlib          12
    string          13
    strlen          60
    to              46      50      59      63
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  SRCHFILE.C - Functions for searching files
**
**  public domain by Bob Stout
**
**  Note: Although this snippet demonstrates some useful techniques, even
**        the fast text searching algorithm used can't provide particularly
**        good performance. Left as an exercise for the user is to perform
**        explicit buffering using fread() rather than fgets() as is used
**        here. See CHBYTES.C in SNIPPETS for how to perform searches in
**        user-managed buffers.
*/

#include <stdlib.h>
#include <string.h>
#include "sniptype.h"
#include "snipfile.h"

/*
**  Allocate a big buffer, use it to buffer a specified stream
*/

static size_t fsetup(FILE *fp, size_t minbuf)
{
      register size_t bufsize;
      register char *buffer;

      /* Allocate the largest buffer we can */

      for (bufsize = 0x4000; bufsize >= minbuf; bufsize >>= 1)
      {
            if (NULL != (buffer = (char *) malloc(bufsize)))
                  break;
      }
      if (NULL == buffer)
            return 0;

      /* Use the buffer to buffer the file */

      if (Success_ == setvbuf(fp, buffer, _IOFBF, bufsize))
            return bufsize;
      else  return 0;
}

/*
**  Search a file for a pattern match (forward)
**
**  Arguments: FILE pointer
**             pattern to search for
**             size of pattern
**             find Nth occurrence
**
**  Returns: -1L if pattern not found
**           -2L in case of error
*/

long ffsearch(FILE *fp, const char *pattern, const size_t size, int N)
{
      long pos = -2L, tempos = 0L;
      char *sbuf, *p;
      size_t skip;
      int ch = 0;

      /* Allocate a search buffer */

      if (NULL == (sbuf = (char *)malloc(size - 1)))
            goto FDONE;

      /* Buffer the file and position us within it */

      if (0 == fsetup(fp, size))
            goto FDONE;
      pos = -1L;
      fseek(fp, 0L, SEEK_SET);

      /* Set up for smart searching */

      if (1 < strlen(pattern) && NULL != (p = strchr(pattern + 1, *pattern)))
            skip = p - (char *)pattern;
      else  skip = strlen(pattern);

      /* Look for the pattern */

      while (EOF != ch)
      {
            if (EOF == (ch = fgetc(fp)))
                  break;
            if ((int)*pattern == ch)
            {
                  tempos = ftell(fp);
                  if (size - 1 > fread(sbuf, sizeof(char), size - 1, fp))
                        goto FDONE;
                  if (Success_ == memcmp(sbuf, &pattern[1], size - 1))
                  {
                        if (0 == --N)
                        {
                              pos = tempos - 1L;
                              goto FDONE;
```

```
                        }
                  }
                  fseek(fp, tempos + skip, SEEK_SET);
            }
      }

      /* Clean up and leave */

FDONE:
      free(sbuf);
      return pos;
}

/*
**  Search a file for a pattern match (backwards)
**
**  Arguments: FILE pointer
**             pattern to search for
**             size of pattern
**             find Nth occurrence
**
**  Returns: -1L if pattern not found
**           -2L in case of error
*/

long rfsearch(FILE *fp, const char *pattern, const size_t size, int N)
{
      long pos = -2L, tempos;
      char *sbuf, *p;
      size_t skip;
      int ch = 0;

      /* Allocate a search buffer */

      if (NULL == (sbuf = (char *)malloc(size - 1)))
            goto RDONE;

      /* Buffer the file and position us within it */

      if (0 == fsetup(fp, size))
            goto RDONE;
      pos = -1L;
      fseek(fp, -1L, SEEK_END);
      tempos = ftell(fp) - strlen(pattern);

      /* Set up for smart searching */

      if (1 < strlen(pattern) && NULL != (p = strrchr(pattern + 1, *pattern)))
            skip = strlen(pattern) - (p - (char *)pattern);
      else  skip = strlen(pattern);

      /* Look for the pattern */

      while (0L <= tempos)
      {
            fseek(fp, tempos, SEEK_SET);
            if (EOF == (ch = fgetc(fp)))
                  break;
            if ((int)*pattern == ch)
            {
                  if (size - 1 <= fread(sbuf, sizeof(char), size - 1, fp))
                  {
                        if (Success_ == memcmp(sbuf, &pattern[1], size - 1))
                        {
                              if (0 == --N)
                              {
                                    pos = tempos;
                                    goto RDONE;
                              }
                        }
                  }
                  tempos -= skip;
            }
            else  --tempos;
      }

      /* Clean up and leave */

RDONE:
      free(sbuf);
      return pos;
}

#ifdef TEST

int main(int argc, char *argv[])
{
      long pos;
      int N = 1;
      size_t size = strlen(argv[1]);
      char buf[256], *fname = "SRCHFILE.C";
      FILE *fp;

      if (2 > argc)
      {
            puts("Usage: SRCHFILE string [N] [file]");
            puts("where: N = find Nth occurrence");
            puts("       If file is specified, N must be given");
            return EXIT_FAILURE;
      }
```

```
201 |
202 |       if (2 < argc)
203 |             N = atoi(argv[2]);
204 |
205 |       if (3 < argc)
206 |             fname = strupr(argv[3]);
207 |
208 |       fp = cant(fname, "r");
209 |       printf("ffsearch(%s, %s) returned %ld\n", fname, argv[1],
210 |             pos = ffsearch(fp, argv[1], size, N));
211 |       fseek(fp, pos, SEEK_SET);
212 |       fgets(buf, 256, fp);
213 |       printf("...which contains \"%s\"\n\n", buf);
214 |       fclose(fp);
215 |
216 |       fp = cant(fname, "rb");
217 |       printf("rfsearch(%s, %s) returned %ld\n", fname, argv[1],
218 |             pos = rfsearch(fp, argv[1], size, N));
219 |       fseek(fp, pos, SEEK_SET);
220 |       fgets(buf, 256, fp);
221 |       printf("...which contains \"%s\"\n\n", buf);
222 |       fclose(fp);
223 |       return EXIT_SUCCESS;
224 | }
225 |
226 | #endif /* TEST */
```

TEXT STATISTICS

```
  6064 characters
   226 lines
```

LEXICAL STATISTICS

```
    18 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
    12 constants [string]
    47 constants [numeric]
```

SYMBOL TABLE

```
    EOF              86      88     157
    EXIT_FAILURE            199
    EXIT_SUCCESS            223
    FDONE            69      74      94     100     109
    FILE             25      59     126     192
    N       59       97     126     165     189     203     210     218
    NULL             34      37      68      80     135     148
    RDONE           136     141     168     179
    SEEK_END        143
    SEEK_SET         76     103     156     211     219
    Success_         42      95     163
    TEST            184
    _IOFBF           42
    argc            186     194     202     205
    argv            186     190     203     206     209     210     217     218
    atoi            203
    buf             191     212     213     220     221
    buffer           28      34      37      42
    bufsize          27      32+     34      42      43
    cant            208     216
    ch               64      86      88      90     131     157     159
    fclose          214     222
    ffsearch         59     210
    fgetc            88     157
    fgets           212     220
    fname           191     206     208     209     216     217
    fp               25      42      59      73      76      88      92      93     103     126     140
           143      144     156     157     161     192     208     210     211     212     214     216
           218      219     220     222
    fread            93     161
    free            110     180
    fseek            76     103     143     156     211     219
    fsetup           25      73     140
    ftell            92     144
    h       16       17
    main            186
    malloc           34      68     135
    memcmp           95     163
    minbuf           25      32
    p       62       80      81     129     148     149
    pattern          59      80+     81      82      90      95     126     144     148+    149+    150
           159      163
    pos              61      75      99     111     128     142     167     181     188     210     211
           218      219
    printf          209     213     217     221
    puts            196     197     198
    rfsearch        126     218
    sbuf             62      68      93      95     110     129     135     161     163     180
    setvbuf          42
    size             59      68      73      93+     95     126     135     140     161+    163     190
           210      218
    size_t           25+     27      59      63     126     130     190
    skip             63      81      82     103     130     149     150     172
    stdlib           16
    strchr           80
    string           17
    strlen           80      82     144     148     149     150     190
    strrchr         148
    strupr          206
    tempos           61      92      99     103     128     144     154     156     167     172     174
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  strcpy() and strcat() work-alikes which allow overlapping buffers.
  5 | */
  6 |
  7 | #include <string.h>
  8 | #include "snip_str.h"
  9 |
 10 | #if defined(__cplusplus) && __cplusplus
 11 |  extern "C" {
 12 | #endif
 13 |
 14 | char *sstrcpy(char *to, char *from)
 15 | {
 16 |     memmove(to, from, 1+strlen(from));
 17 |     return to;
 18 | }
 19 |
 20 | char *sstrcat(char *to, char *from)
 21 | {
 22 |     sstrcpy(to + strlen(to), from);
 23 |     return to;
 24 | }
 25 |
 26 | #if defined(__cplusplus) && __cplusplus
 27 |  }
 28 | #endif
```

TEXT STATISTICS

```
   490 characters
    28 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     2 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    __cplusplus             10+     26+
    defined         10      26
    from            14      16+     20      22
    h       7
    memmove         16
    sstrcat         20
    sstrcpy         14      22
    string           7
    strlen          22
    to              14      16      17      20      22+     23
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  sstrdel() - Delete multiple substrings
**
**  public domain by Shamim Islam
**
**  Usage: sstrdel(char * s,char * del0,del1...deln)
**
**  Remarks: sstrdel takes a string s, and removes all occurrences of
**           the substrings del0, del1, ... deln
**
**  Return:  sstrdel returns a pointer to the string s, unless s is NULL.
**           sstrdel will return a NULL for this exception.
**
**  Comment: Use sstrdel to remove a list of substrings from a string.
**
**           sstrdel matches the largest substring for deletion, if more than
**           one substring matches a particular portion of the string s.
**
**  NOTE:    The last element in the variable substring list MUST be NULL
**           or your program will have a high likelihood of hanging.
*/

#include <stdarg.h>
#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *sstrdel(char *s, ...)
{
      /* Find out how many arguments are present */

      int c = 0;
      va_list ap, hold;

      if (s == NULL)
            return NULL;
      va_start(ap, s);
      memcpy(&hold, &ap, sizeof(va_list));
      while (va_arg(ap, char*) != NULL)
            c++;
      va_end(ap);
      if (c)
      {
            /* Assign pointers  */

            char *r = s,*n = s;
            char *p;
            int len, i;

            /* Copy next character to result */
            /* And then check for matches if */
            /* not at end of string          */

            while ((*r = *n) != 0)
            {
                  int l = 0;

                  /* Substitute for va_start(ap,s) */

                  memcpy(&ap, &hold, sizeof(va_list));
                  for (i = 0; i < c; i++)
                  {
                        /* Initialise the pointer and the length    */

                        len = strlen(p = va_arg(ap, char*));

                        /* Compare ONLY if we haven't found one yet */
                        /* or if this one is bigger than the one we */
                        /* found AND this arg has a length > 0      */

                        if(len > 0 && (l == 0 || len> l) &&
                              strncmp(n, p, len) == 0)
                        {
                              l = len;
                        }
                  }
                  va_end(ap);
                  if (l)
                        n += l;
                  else  n++, r++;
            }
      }
      return s;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

main()
{
```

```
 101 |       char *testr = "The quick brown fox jumps over the lazy dog.";
 102 |       char *s1 = "quick", *s2 = "over", *s3 = "lazy";
 103 |       char *s4 = "he", *s5 = "xyz";
 104 |
 105 |       printf("sstrdel(\"%s\",\"%s\",\"%s\",\"%s\")\n returned ",
 106 |              testr, s1, s2, s3);
 107 |       printf("\"%s\"\n", sstrdel(testr, s1, s2, s3, NULL));
 108 |
 109 |       printf("sstrdel(\"%s\",\"%s\",\"%s\")\n returned ",
 110 |              testr, s4, s5);
 111 |       printf("\"%s\"\n", sstrdel(testr, s4, s5, NULL));
 112 |       return 0;
 113 | }
 114 |
 115 | #endif /* TEST */
```

TEXT STATISTICS

```
  3230 characters
   115 lines
```

LEXICAL STATISTICS

```
    13 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
    12 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    NULL            40      41      44     107     111
    TEST            95
    __cplusplus             29+     91+
    ap              38      42      43      44      46      65      70      82
    c       37      45      47      66
    defined         29      91
    h       25      26      97
    hold            38      43      65
    i       53      66+
    l       61      76+     79      83      84
    len             53      70      76+     77      79
    main            99
    memcpy          43      65
    n       51      59      77      84      85
    p       52      70      77
    printf         105     107     109     111
    r       51      59      85
    s       33      40      42      51+     88
    s1             102     106     107
    s2             102     106     107
    s3             102     106     107
    s4             103     110     111
    s5             103     110     111
    sstrdel         33     107     111
    stdarg          25
    stdio           97
    string          26
    strlen          70
    strncmp         77
    testr          101     106     107     110     111
    va_arg          44      70
    va_end          46      82
    va_list         38      43      65
    va_start        42
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    STK_BLOB.c      Stack of Blobs. Binary Large OBjects.

                    A.Reitsma, Delft, The Netherlands.
                    v0.10  94-07-03  Public Domain.

    Stack for variable sized items. Implemented by making a stack of
    'Blob Descriptors' and duplicating the item in malloc'ed memory.
    Localizing the stack number storage required interface functions
    to the stack module.
    The Pop function has two 'modes': one just returns the size of the
    blob; the other actually pops the blob.
    WARNING: This version is not really tested!
---------------------------------------------------------------------- */

#include <string.h>
#include "stk_defs.h"
#include "stack.h"
#include "stk_blob.h"

#define ERR_MEMORY   -1
#define NO_PROBLEMS  0
#define Error(x)     (x)

struct BlobDesc
{
    void * data;
    int size;
};

int StackBlobCreate( void )
{
    return StackCreate( sizeof( struct BlobDesc ));
}

int PushBlob( int Stack, void * BlobSource, unsigned int BlobSize )
{
    struct BlobDesc Blob;

    Blob.size = BlobSize;

    /* get storage space for the blob
    */
    Blob.data = MALLOC( BlobSize, char );
    if( NULL == Blob.data )
    {
        return ERR_MEMORY ;
    }

    memcpy( Blob.data, BlobSource, BlobSize ); /* duplicate the blob */

    if( ERR_MEMORY == Push( Stack, & Blob )) /* push descriptor */
    {
        FREE( Blob.data ); /* no need to cause memory leaks ... */
        return ERR_MEMORY ;
    }

    return NO_PROBLEMS ;
}

unsigned int PopBlob( int Stack, void * BlobDestination )
{
    struct BlobDesc Blob ;

    Pop( Stack, & Blob );

    if( NULL == BlobDestination )
        StackUnpop( Stack );
    else
    {
        memcpy( BlobDestination, Blob.data, Blob.size );
        FREE( Blob.data );
    }

    return Blob.size ;
}

/* ==== STK_BLOB.c end ================================================ */
```

TEXT STATISTICS

```
  2071 characters
    80 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     3 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    Blob            40      42      46      47      52      54      56      65      67      73+     74
            77
    BlobDesc        27      35      40      65
    BlobDestination         63      69      73
    BlobSize        38      42      46      52
    BlobSource              38      52
    ERR_MEMORY              23      49      54      57
    Error           25
    FREE            56      74
    MALLOC          46
    NO_PROBLEMS             24      60
    NULL            47      69
    Pop             67
    PopBlob         63
    Push            54
    PushBlob        38
    Stack           38      54      63      67      70
    StackBlobCreate         33
    StackCreate             35
    StackUnpop              70
    data            29      46      47      52      56      73      74
    h       18
    memcpy          52      73
    size            30      42      73      77
    string          18
    x       25+
```

```
/* +++Date last modified: 05-Jul-1997 */

/* ======================================================================
    STK_BLOB.h      Stack of Blobs. Binary Large OBjects.

                    A.Reitsma, Delft, The Netherlands.
                    v0.10  94-07-03  Public Domain.

    This module is essentially an interface to the generic Stack module.
    The size of the items is however variable.
    This module is still VERY experimental.
    A separate FAR version required ???
---------------------------------------------------------------------- */


#ifndef STK_BLOB_H
#define STK_BLOB_H

int  StackBlobCreate( void );

int  PushBlob( int Stack, void * BlobSource, unsigned int BlobSize );
                /* Duplicates contents of BlobSource on a stack         */

unsigned int PopBlob( int Stack, void * BlobDestination );
                /* Returns blobsize. If BlobDestination == NULL that's  */
                /* all it does. I.e: it is NOT a real Pop.              */
                /* Not NULL: Blob data is put into the indicated space, */
                /* i.e. it is a 'real' Pop.                             */
                /* Purpose: giving caller opportunity to allocate space */

#endif
/* ==== STK_BLOB.h end ================================================ */
```

TEXT STATISTICS

```
  1330 characters
    32 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    BlobDestination         24
    BlobSize        21
    BlobSource              21
    PopBlob         24
    PushBlob        21
    STK_BLOB_H              16      17
    Stack           21      24
    StackBlobCreate         19
```

```
/* +++Date last modified: 05-Jul-1997 */

/*  ======================================================================
    DEFINES.h       Standard definitions etc.
                    For simplification or for debugging substitution.

                    A.Reitsma, Delft, Nederland.
                    v0.23  94-07-02
---------------------------------------------------------------------- */

#ifndef STK_DEFS_H
#define STK_DEFS_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char       byte;
typedef unsigned short      byte2;
typedef unsigned long       byte4;
typedef unsigned int        word;

#define MALLOC(size,type)   (type *) malloc( (size) * sizeof( type ))
#define FREE(mem)           free( mem )
#define CALLOC(size,type)   (type *) calloc( (size), sizeof( type))

#ifdef DEBUG_NPA

extern void * NPAdata ;     /* Declare it in the main() file !!!       */

#if sizeof( void * ) != sizeof( void far * )   /* small data models */
#define NPAsize             0x40
#else
#define NPAsize             0x100 * sizeof( void far * )
#endif                      /* protect interrupt vectors for
                               large data models                       */

#define NPAinit()           NPAdata = malloc( NPAsize );      \
                            memcpy( NPAdata, NULL, NPAsize )
                            /* call NPAinit as first function in main() */

#define NPAcheck()          if( memcmp( NPAdata, NULL, NPAsize ))        \
                            {   printf( "\aNPA before line %d"           \
                                    " in file %s", __LINE__, __FILE__ ); \
                                memcpy( NULL, NPAdata, NPAsize );        \
                            }
#if 0
#define return(x)           { NPAcheck(); return(x); }
                            /* does NOT work for format "return XXX ;"  */
#else
#define return              NPAcheck(); return
                            /* does also work for format "return XXX ;" */
#endif                      /* but not for:
                                  if(...) return ...;  without { }      */
                            /* that causes warnings of unreachable code */
#pragma warn +rch           /* when switched on with this pragma ...    */
                            /* TC2.0, BC3.1 */
#else

#define NPAinit()
#define NPAcheck()

#endif

#endif /* STK_DEFS_H */

/*  === DEFINES.h end ================================================= */
```

```
TEXT STATISTICS

  2539 characters
    67 lines

LEXICAL STATISTICS

    14 comments [std-C]
     0 comments [C++]
    27 preprocessor instructions
     0 constants [character]
     2 constants [string]
     3 constants [numeric]

SYMBOL TABLE

    CALLOC          25
    DEBUG_NPA       27
    FREE            24
    MALLOC          23
    NPAcheck        42      48      51      61
    NPAdata         29      38      39      42      45
    NPAinit         38      60
    NPAsize         32      34      38      39      42      45
    NULL            39      42      45
    STK_DEFS_H              11      12
    __FILE__        44
    __LINE__        44
    byte            18
    byte2           19
    byte4           20
    calloc          25
    far             31      34
    free            24
    h       14      15      16
    malloc          23      38
    mem             24+
    memcmp          42
    memcpy          39      45
    printf          43
    rch             56
    size            23+     25+
    stdio           14
    stdlib          15
    string          16
    type            23+     25+
    warn            56
    word            21
    x       48+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* ======================================================================
  4 |     STK_STR.h       Stack of Strings based on generic stack of Blobs.
  5 |
  6 |                     A.Reitsma, Delft, The Netherlands.
  7 |                     v0.10  94-07-02  Public Domain.
  8 |
  9 |                     This module has no related .C file.
 10 |                     Modules STK_BLOB and STACK are required.
 11 | ___---------------------------------------------------------------- */
 12 |
 13 | #ifndef STK_STR_H
 14 | #define STK_STR_H
 15 |
 16 | #include <string.h>
 17 | #include "stk_blob.h"
 18 |
 19 | #define StackStringCreate()       StackBlobCreate()
 20 | #define PushString(Stack,Str)     PushBlob( Stack, Str, strlen( Str ) +1 )
 21 | #define PopString(Stack,Str)      PopBlob( Stack, Str )
 22 |
 23 | #endif
 24 | /* ==== STK_STR.c end ============================================== */
```

TEXT STATISTICS

```
   859 characters
    24 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     1 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    PopBlob         21
    PopString       21
    PushBlob        20
    PushString              20
    STK_STR_H       13      14
    Stack           20+     21+
    StackBlobCreate         19
    StackStringCreate       19
    Str             20+     21+
    h       16
    string          16
    strlen          20
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  stptok() -- public domain by Ray Gardner, modified by Bob Stout
**
**   You pass this function a string to parse, a buffer to receive the
**   "token" that gets scanned, the length of the buffer, and a string of
**   "break" characters that stop the scan.  It will copy the string into
**   the buffer up to any of the break characters, or until the buffer is
**   full, and will always leave the buffer null-terminated.  It will
**   return a pointer to the first non-breaking character after the one
**   that stopped the scan.
*/

#include <string.h>
#include <stdlib.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *stptok(const char *s, char *tok, size_t toklen, char *brk)
{
      char *lim, *b;

      if (!*s)
            return NULL;

      lim = tok + toklen - 1;
      while ( *s && tok < lim )
      {
            for ( b = brk; *b; b++ )
            {
                  if ( *s == *b )
                  {
                        *tok = 0;
                        for (++s, b = brk; *s && *b; ++b)
                        {
                              if (*s == *b)
                              {
                                    ++s;
                                    b = brk;
                              }
                        }
                        return (char *)s;
                  }
            }
            *tok++ = *s++;
      }
      *tok = 0;
      return (char *)s;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

main(int argc, char *argv[])
{
      char *ptr, buf[256];

      if (3 > argc)
      {
            puts("Usage: STPTOK \"string\" \"token_string\"");
            return EXIT_FAILURE;
      }
      else  ptr = argv[1];
      do
      {
            ptr = stptok(ptr, buf, sizeof(buf), argv[2]);
            printf("stptok(\"%s\", \"%s\")\n  buf: \"%s\"\n  "
                  "returned: \"%s\"\n", argv[1], argv[2], buf, ptr);
      } while (ptr && *ptr);
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2173 characters
    82 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     5 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            70
    EXIT_SUCCESS            79
    NULL            28
    TEST            59
    __cplusplus             19+     55+
    argc            63      67
    argv            63      72      75      77+
    b       25      33+     35      38+     40      43
    brk             23      33      38      43
    buf             65      75+     77
    defined         19      55
    h       15      16      61
    lim             25      30      31
    main            63
    printf          76
    ptr             65      72      75+     77      78+
    puts            69
    s       23      27      31      35      38+     40      42      46      49      52
    size_t          23
    stdio           61
    stdlib          16
    stptok          23      75
    string          15
    tok             23      30      31      37      49      51
    toklen          23      30
```

```
/* +++Date last modified: 05-Jul-1997 */

//
// Simple string class
// Public domain
//
// Written by david nugent
// davidn@csource.pronet.com
// 3:632/348@fidonet
//

# if !defined( _str_h )
# define _str_h 1
# if !defined( _Bool_defined )
enum Bool { False, True };
# endif

# if defined( __BORLANDC__ ) && ( __BORLANDC__ <= 0x3000 )
# define PLACEMENT_NEW_BUG
# define SIGNED_CHAR_BUG
# endif

// Define to 0 for much smaller class
#define VIRTUAL_DESTRUCTOR  0
#if VIRTUAL_DESTRUCTOR
#define __VIRTUAL   virtual
#else
#define __VIRTUAL
#endif

struct refstr
{
    short _size;
    short _length;
    short _refs;
    unsigned short _flags;

    enum rsflags { ICASE=1 };

    refstr (short length, short size, unsigned short flgs =0)
        : _length(length), _size(size), _refs(1), _flags(flgs)
        {}
    ~refstr (void) {}
    unsigned short flags() const
    {
        return _flags;
    }
    void setf(unsigned short f)
    {
        _flags = (unsigned short)(_flags | f);
    }
    void resetf(unsigned short f)
    {
        _flags = (unsigned short)(_flags & ~f);
    }
# if !defined( PLACEMENT_NEW_BUG )
    void * operator new(unsigned sz, short allocsz);
# endif
    char * ptr (void)
    {
        return ((char *)this) + sizeof(refstr);
    }
};




class str
{

  public:

        // constructors/destructors

    str (void);
    str (char const * s, short len =-1);
    str (unsigned char const * s, short len =-1);
# if !defined( SIGNED_CHAR_BUG )
    str (signed char const * s, short len =-1);
# endif
    str (int val, int radix =10);
    str (unsigned int val, int radix =10);
    str (short val, int radix =10);
    str (unsigned short val, int radix =10);
    str (long val, int radix =10);
    str (unsigned long val, int radix =10);
    str (char c);
    str (unsigned char c);
# if !defined( SIGNED_CHAR_BUG )
    str (signed char c);
# endif
    str (str const & s);
    __VIRTUAL ~str (void);

    str & clear(void);

        // assignment

    str & operator= (str const & s);
    str & operator= (char const * s);
```

```
101 |     str & operator= (char c);
102 |
103 |     str & operator= (unsigned char const * s)
104 |     {
105 |         return operator= ((char const *)s);
106 |     }
107 | # if !defined( SIGNED_CHAR_BUG )
108 |     str & operator= (signed char const * s)
109 |     {
110 |         return operator= ((char const *)s);
111 |     }
112 | # endif
113 |
114 |         // primitive members
115 |
116 |     short length (void) const
117 |     {
118 |         return strdata->_length;
119 |     }
120 |
121 |     short size (void) const
122 |     {
123 |         return strdata->_size;
124 |     }
125 |
126 |         // operators
127 |
128 |     str & operator<< (char const * s);              // concatenate
129 |     str & operator<< (unsigned char const * s);
130 | # if !defined( SIGNED_CHAR_BUG )
131 |     str & operator<< (signed char const * s);
132 | # endif
133 |     str & operator<< (str const & s);
134 |     str & operator<< (int val);
135 |     str & operator<< (unsigned int val);
136 |     str & operator<< (short val);
137 |     str & operator<< (unsigned short val);
138 |     str & operator<< (long val);
139 |     str & operator<< (unsigned long val);
140 |     str & operator<< (char c);
141 |     str & operator<< (unsigned char c);
142 | # if !defined( SIGNED_CHAR_BUG )
143 |     str & operator<< (signed char c);
144 | # endif
145 |
146 |     char const & operator[] (short pos) const;
147 |     char & operator[] (short pos);
148 |
149 |     char * c_ptr (void) const            // not necessarily NUL terminated!
150 |     {                                                 // Use with caution...
151 |         return strdata->ptr();
152 |     }
153 |     char const * c_str (void) const    // return char*
154 |     {
155 |         char * buf = c_ptr();
156 |         buf[strdata->_length] = 0;
157 |         return buf;
158 |     }
159 |     unsigned char const * u_str (void) const
160 |     {
161 |         return (unsigned char const *)c_str();
162 |     }
163 | # if !defined( SIGNED_CHAR_BUG )
164 |     signed char const * s_str (void) const
165 |     {
166 |         return (signed char const *)c_str();
167 |     }
168 | # endif
169 |
170 |     int copy(char * dest, short maxlen =-1) const;
171 |
172 |         // manipulators
173 |
174 |     short insert (short pos, char const * s, short len =-1);
175 |     short insert (short pos, str const & s)
176 |     {
177 |         return insert (pos, s.c_ptr(), s.length());
178 |     }
179 |     short insert (short pos, unsigned char const * s, short len =-1)
180 |     {
181 |         return insert (pos, (char const *)s, len);
182 |     }
183 | # if !defined( SIGNED_CHAR_BUG )
184 |     short insert (short pos, signed char const * s,
185 |                   short len =-1)
186 |     {
187 |         return insert (pos, (char const *)s, len);
188 |     }
189 | # endif
190 |     short insert (short pos, char c)
191 |     {
192 |         return insert (pos, &c, 1);
193 |     }
194 |     short insert (short pos, unsigned char c)
195 |     {
196 |         return insert (pos, (char const *)&c, 1);
197 |     }
198 | # if !defined( SIGNED_CHAR_BUG )
199 |     short insert (short pos, signed char c)
200 |     {
```

```
201 |         return insert (pos, (char const *)&c, 1);
202 |     }
203 | # endif
204 |
205 |     short remove (short pos =0, short len =-1);
206 |     short replace (short pos, char const * s, short clen =-1, short len =-1);
207 |     short replace (short pos, str & s, short clen =-1)
208 |     {
209 |         return replace (pos, s.c_ptr(), clen, s.length());
210 |     }
211 |     short replace (short pos, unsigned char const * s, short clen =-1, short len =-1)
212 |     {
213 |         return replace (pos, (char const *)s, clen, len);
214 |     }
215 | # if !defined( SIGNED_CHAR_BUG )
216 |     short replace (short pos, signed char const * s, short clen =-1, short len =-1)
217 |     {
218 |         return replace (pos, (char const *)s, clen, len);
219 |     }
220 | # endif
221 |     short replace (short pos, char c, short clen =-1)
222 |     {
223 |         return replace (pos, &c, clen, 1);
224 |     }
225 |     short replace (short pos, unsigned char c, short clen =-1)
226 |     {
227 |         return replace (pos, (char const *)&c, clen, 1);
228 |     }
229 | # if !defined( SIGNED_CHAR_BUG )
230 |     short replace (short pos, signed char c, short clen =-1)
231 |     {
232 |         return replace (pos, (char const *)&c, clen, 1);
233 |     }
234 | # endif
235 |
236 |     str & left (short len, char padch =' ');
237 |     str & right (short len, char padch =' ');
238 |     str & mid (short pos, short len, char padch =' ');
239 |
240 |     str substr(short start, short len =-1) const;
241 |
242 |     short removech (char const * clist ="\r\n");
243 |     short countch (char const * clist);
244 |
245 |     Bool operator== (str const & s) const
246 |     {
247 |         return (_compare(s) == 0) ? True : False;
248 |     }
249 |     Bool operator== (char const * s) const
250 |     {
251 |         return (_compare(s) == 0) ? True : False;
252 |     }
253 |     Bool operator== (unsigned char const * s) const
254 |     {
255 |         return (_compare(s) == 0) ? True : False;
256 |     }
257 | # if !defined( SIGNED_CHAR_BUG )
258 |     Bool operator== (signed char const * s) const
259 |     {
260 |         return (_compare(s) == 0) ? True : False;
261 |     }
262 | #endif
263 |
264 |     Bool operator!= (str const & s) const
265 |     {
266 |         return (_compare(s) != 0) ? True : False;
267 |     }
268 |     Bool operator!= (char const * s) const
269 |     {
270 |         return (_compare(s) != 0) ? True : False;
271 |     }
272 |     Bool operator!= (unsigned char const * s) const
273 |     {
274 |         return (_compare(s) != 0) ? True : False;
275 |     }
276 | # if !defined( SIGNED_CHAR_BUG )
277 |     Bool operator!= (signed char const * s) const
278 |     {
279 |         return (_compare(s) != 0) ? True : False;
280 |     }
281 | #endif
282 |
283 |     Bool operator< (str const & s) const
284 |     {
285 |         return (_compare(s) < 0) ? True : False;
286 |     }
287 |     Bool operator< (char const * s) const
288 |     {
289 |         return (_compare(s) < 0) ? True : False;
290 |     }
291 |     Bool operator< (unsigned char const * s) const
292 |     {
293 |         return (_compare(s) < 0) ? True : False;
294 |     }
295 | # if !defined( SIGNED_CHAR_BUG )
296 |     Bool operator< (signed char const * s) const
297 |     {
298 |         return (_compare(s) < 0) ? True : False;
299 |     }
300 | #endif
```

```
301 |
302 |     Bool operator<= (str const & s) const
303 |     {
304 |         return (_compare(s) <= 0) ? True : False;
305 |     }
306 |     Bool operator<= (char const * s) const
307 |     {
308 |         return (_compare(s) <= 0) ? True : False;
309 |     }
310 |     Bool operator<= (unsigned char const * s) const
311 |     {
312 |         return (_compare(s) <= 0) ? True : False;
313 |     }
314 | # if !defined( SIGNED_CHAR_BUG )
315 |     Bool operator<= (signed char const * s) const
316 |     {
317 |         return (_compare(s) <= 0) ? True : False;
318 |     }
319 | #endif
320 |
321 |     Bool operator> (str const & s) const
322 |     {
323 |         return (_compare(s) > 0) ? True : False;
324 |     }
325 |     Bool operator> (char const * s) const
326 |     {
327 |         return (_compare(s) > 0) ? True : False;
328 |     }
329 |     Bool operator> (unsigned char const * s) const
330 |     {
331 |         return (_compare(s) > 0) ? True : False;
332 |     }
333 | # if !defined( SIGNED_CHAR_BUG )
334 |     Bool operator> (signed char const * s) const
335 |     {
336 |         return (_compare(s) > 0) ? True : False;
337 |     }
338 | #endif
339 |
340 |     Bool operator>= (str const & s) const
341 |     {
342 |         return (_compare(s) >= 0) ? True : False;
343 |     }
344 |     Bool operator>= (char const * s) const
345 |     {
346 |         return (_compare(s) >= 0) ? True : False;
347 |     }
348 |     Bool operator>= (unsigned char const * s) const
349 |     {
350 |         return (_compare(s) >= 0) ? True : False;
351 |     }
352 | # if !defined( SIGNED_CHAR_BUG )
353 |     Bool operator>= (signed char const * s) const
354 |     {
355 |         return (_compare(s) >= 0) ? True : False;
356 |     }
357 | #endif
358 |
359 |     int compare (str const & s) const
360 |     {
361 |         return _compare(s);
362 |     }
363 |     int compare (char const * s) const
364 |     {
365 |         return _compare(str(s));
366 |     }
367 |     int compare (unsigned char const * s) const
368 |     {
369 |         return _compare(str(s));
370 |     }
371 | # if !defined( SIGNED_CHAR_BUG )
372 |     int compare (signed char const * s) const
373 |     {
374 |         return _compare(str(s));
375 |     }
376 | #endif
377 |
378 |     short strstr (str const & s) const
379 |     {
380 |         return _strstr(s);
381 |     }
382 |     short strstr (char const * s) const
383 |     {
384 |         return _strstr(str(s));
385 |     }
386 |     short strstr (unsigned char const * s) const
387 |     {
388 |         return _strstr(str(s));
389 |     }
390 | # if !defined( SIGNED_CHAR_BUG )
391 |     short strstr (signed char const * s) const
392 |     {
393 |         return _strstr(str(s));
394 |     }
395 | #endif
396 |
397 |     void setflags (unsigned short flags);
398 |     void resetflags (unsigned short flags);
399 |
400 |     void setcase (Bool s =True)
```

```
401 |     {
402 |         if (s)
403 |             setflags(refstr::ICASE);
404 |         else
405 |             resetflags(refstr::ICASE);
406 |     }
407 |
408 |     static void setdefaultcase (Bool s = True)
409 |     {
410 |         default_flags = (s == False)
411 |                         ? (unsigned short)(default_flags | refstr::ICASE)
412 |                         : (unsigned short)(default_flags & ~refstr::ICASE);
413 |     }
414 |
415 |   protected:
416 |
417 |     static unsigned short default_flags;
418 |     refstr * strdata;
419 |
420 |     // Check to see if big enough for size
421 |     int _chksize (short sz =0);
422 |     int _concat (char const * s, short len =-1);
423 |     int _concat (str const & s)
424 |     {
425 |         return _concat (s.c_ptr(), s.length());
426 |     }
427 |     int _concat (char ch)
428 |     {
429 |         return _concat (&ch, 1);
430 |     }
431 |     int _concat (unsigned char const * s, short len =-1)
432 |     {
433 |         return _concat ((char const *)s, len);
434 |     }
435 | # if !defined( SIGNED_CHAR_BUG )
436 |     int _concat (signed char const * s, short len =-1)
437 |     {
438 |         return _concat ((char const *)s, len);
439 |     }
440 | # endif
441 |
442 |     int _compare (str const s) const;
443 |     short _strstr (str const s) const;
444 |
445 |         // Common constructor
446 |
447 |     void _strinit (char const * s =0, short slen =0,
448 |                    short siz =-1, unsigned short flgs =default_flags);
449 |     void _strinit (unsigned long val, Bool positive, int radix);
450 |
451 | };
452 |
453 |
454 | inline str
455 | left (str s, short len, char padch =' ')
456 | {
457 |     return s.left(len, padch);
458 | }
459 |
460 | inline str
461 | right (str s, short len, char padch =' ')
462 | {
463 |     return s.right(len, padch);
464 | }
465 |
466 | inline str
467 | mid (str s, short pos, short len, char padch =' ')
468 | {
469 |     return s.mid(pos, len, padch);
470 | }
471 |
472 | inline int
473 | compare(str s, str b)
474 | {
475 |     return s.compare(b);
476 | }
477 |
478 | inline int
479 | compare(str s, char const * b)
480 | {
481 |     return s.compare(b);
482 | }
483 |
484 | inline int
485 | compare(str s, unsigned char const * b)
486 | {
487 |     return s.compare(b);
488 | }
489 |
490 | # if !defined( SIGNED_CHAR_BUG )
491 | inline int
492 | compare(str s, signed char const * b)
493 | {
494 |     return s.compare(b);
495 | }
496 | # endif
497 |
498 | class ostream;
499 | class istream;
500 |
```

```
 501 | ostream & operator<< (ostream & os, str const & s);
 502 | istream & operator>> (istream & is, str & s);
 503 |
 504 | # endif


TEXT STATISTICS

 12325 characters
   504 lines

LEXICAL STATISTICS

     1 comments [std-C]
    20 comments [C++]
    14 preprocessor instructions
     6 constants [character]
     1 constants [string]
    71 constants [numeric]

SYMBOL TABLE

    Bool             15     245     249     253     258     264     268     272     277     283     287
           291     296     302     306     310     315     321     325     329     334     340     344
           348     353     400     408     449
    False            15     247     251     255     260     266     270     274     279     285     289
           293     298     304     308     312     317     323     327     331     336     342     346
           350     355     410
    ICASE            38     403     405     411     412
    PLACEMENT_NEW_BUG        19      56
    SIGNED_CHAR_BUG          20      78      89     107     130     142     163     183     198     215
           229     257     276     295     314     333     352     371     390     435     490
    True             15     247     251     255     260     266     270     274     279     285     289
           293     298     304     308     312     317     323     327     331     336     342     346
           350     355     400     408
    VIRTUAL_DESTRUCTOR               24      25
    _Bool_defined            14
    __BORLANDC__             18+
    __VIRTUAL        26      28      93
    _chksize        421
    _compare        247     251     255     260     266     270     274     279     285     289     293
           298     304     308     312     317     323     327     331     336     342     346     350
           355     361     365     369     374     442
    _concat         422     423     425     427     429     431     433     436     438
    _flags           36      41      46      50+     54+
    _length          34      41     118     156
    _refs            35      41
    _size            33      41     123
    _str_h           12      13
    _strinit        447     449
    _strstr         380     384     388     393     443
    allocsz          57
    b       473     475     479     481     485     487     492     494
    buf             155     156     157
    c        87      88      90     101     140     141     143     190     192     194     196     199
           201     221     223     225     227     230     232
    c_ptr           149     155     177     209     425
    c_str           153     161     166
    ch              427     429
    clear            95
    clen            206     207     209     211     213     216     218     221     223     225     227
           230     232
    clist           242     243
    compare         359     363     367     372     473     475     479     481     485     487     492
           494
    copy            170
    countch         243
    default_flags            410     411     412     417     448
    define           13      19      20
    defined          12      14      18      56      78      89     107     130     142     163     183
           198     215     229     257     276     295     314     333     352     371     390     435
           490
    dest            170
    endif            16      21      58      80      91     112     132     144     168     189     203
           220     234     440     496     504
    f        48      50      52      54
    flags            44     397     398
    flgs             40      41     448
    insert          174     175     177     179     181     184     187     190     192     194     196
           199     201
    is              502
    istream         499     502+
    left            236     455     457
    len              76      77      79     174     179     181     185     187     205     206     211
           213     216     218     236     237     238     240     422     431     433     436     438
           455     457     461     463     467     469
    length           40      41     116     177     209     425
    maxlen          170
    mid             238     467     469
    os              501
    ostream         498     501+
    padch           236     237     238     455     457     461     463     467     469
    pos             146     147     174     175     177     179     181     184     187     190     192
           194     196     199     201     205     206     207     209     211     213     216     218
           221     223     225     227     230     232     238     467     469
    positive        449
    ptr              59     151
    radix            81      82      83      84      85      86     449
    refstr           31      40      43      61     403     405     411     412     418
    remove          205
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STR27SEG.C - Convert numeric strings to 7-segment strings.
**
**  Public domain by Bob Stout
**
**  Input:  A string (NUL-delimited char array) containing only digits
**          ('0' - '9' chars).
**
**  Output: The same string with each digit converted to a 7-segment
**          representation. Returns NULL on error.
*/

#include <stdio.h>
#include <ctype.h>
#include "numcnvrt.h"

#define CAST(new_type,old_object) (*((new_type *)&old_object))
#define DISP(str) fputs((str), stdout)

/*
**  Define the bit significance
**
**     a
**    ---
**   |   |
**  f|   |b
**   | g |
**    ---
**   |   |
**  e|   |c
**   |   |
**    ---
**     d
*/

struct Seg7disp Seg7digits[10] = {
      { 1, 1, 1, 1, 1, 1, 0 },      /* 0 */
      { 0, 1, 1, 0, 0, 0, 0 },      /* 1 */
      { 1, 1, 0, 1, 1, 0, 1 },      /* 2 */
      { 1, 1, 1, 1, 0, 0, 1 },      /* 3 */
      { 0, 1, 1, 0, 0, 1, 1 },      /* 4 */
      { 1, 0, 1, 1, 0, 1, 1 },      /* 5 */
      { 1, 0, 1, 1, 1, 1, 1 },      /* 6 */
      { 1, 1, 1, 0, 0, 0, 0 },      /* 7 */
      { 1, 1, 1, 1, 1, 1, 1 },      /* 8 */
      { 1, 1, 1, 1, 0, 1, 1 }       /* 9 */
};

char *str27seg(char *string)
{
      char *str;
      int ch;

      for (str = string ; *str; ++str)
      {
            if (!isdigit(*str))
                  return NULL;
            ch = CAST(int, Seg7digits[*str - '0']);
            *str = (char)(ch & 0xff);
      }
      return string;
}

#ifdef TEST

main()
{
      char tstrng[] = "0123456789", *segs;

      printf("str27seg() returned %p", segs = str27seg(tstrng));
      if (segs)
      {
            char *ptr;
            struct Seg7disp ch;
            int i;

            puts(" & produced:\n");
            for (ptr = segs ; *ptr; ++ptr)
            {
                  ch = CAST(struct Seg7disp, *ptr);
                  if (ch.seg_a)
                        DISP(" --- ");
                  else  DISP("     ");
                  DISP("  ");
            }
            puts("");
            for (i = 0; i < 3; ++i)
            {
                  for (ptr = segs ; *ptr; ++ptr)
                  {
                        ch = CAST(struct Seg7disp, *ptr);
                        if (ch.seg_f)
                              DISP("|   ");
                        else  DISP("    ");
                        if (ch.seg_b)
                              DISP("|");
                        else  DISP(" ");
                        DISP("  ");
```

```
101 |                   }
102 |                   puts("");
103 |             }
104 |             for (ptr = segs ; *ptr; ++ptr)
105 |             {
106 |                   ch = CAST(struct Seg7disp, *ptr);
107 |                   if (ch.seg_g)
108 |                         DISP(" --- ");
109 |                   else  DISP("     ");
110 |                   DISP("  ");
111 |             }
112 |             puts("");
113 |             for (i = 0; i < 3; ++i)
114 |             {
115 |                   for (ptr = segs ; *ptr; ++ptr)
116 |                   {
117 |                         ch = CAST(struct Seg7disp, *ptr);
118 |                         if (ch.seg_e)
119 |                               DISP("|   ");
120 |                         else  DISP("    ");
121 |                         if (ch.seg_c)
122 |                               DISP("|");
123 |                         else  DISP(" ");
124 |                         DISP("  ");
125 |                   }
126 |                   puts("");
127 |             }
128 |             for (ptr = segs ; *ptr; ++ptr)
129 |             {
130 |                   ch = CAST(struct Seg7disp, *ptr);
131 |                   if (ch.seg_d)
132 |                         DISP(" --- ");
133 |                   else  DISP("     ");
134 |                   DISP("  ");
135 |             }
136 |             puts("");
137 |       }
138 |       else  puts("\n");
139 |
140 |       return 0;
141 | }
142 |
143 | #endif /* TEST */
```

TEXT STATISTICS

```
  3846 characters
   143 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     1 constants [character]
    29 constants [string]
    77 constants [numeric]
```

SYMBOL TABLE

```
    CAST            19      60      82      93     106     117     130
    DISP            20      84      85      86      95      96      98      99     100     108     109
           110     119     120     122     123     124     132     133     134
    NULL            59
    Seg7digits              38      60
    Seg7disp        38      76      82      93     106     117     130
    TEST            66
    ch              54      60      61      76      82      83      93      94      97     106     107
           117     118     121     130     131
    ctype           16
    fputs           20
    h       15      16
    i       77      89+    113+
    isdigit         58
    main            68
    new_type        19+
    old_object              19+
    printf          72
    ptr             75      80+     82      91+     93     104+    106     115+    117     128+    130
    puts            79      88     102     112     126     136     138
    seg_a           83
    seg_b           97
    seg_c          121
    seg_d          131
    seg_e          118
    seg_f           94
    seg_g          107
    segs            70      72      73      80      91     104     115     128
    stdio           15
    stdout          20
    str             20+     53      56+     58      60      61
    str27seg        51      72
    string          51      56      63
    tstrng          70      72
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  strat.c  10-5-91  Robert Mashlan, public domain
**
**   Interface functions to DOS 3.0+ set allocation strategy
**   and get allocation strategy functions via int 21h,
**   function 58h.
**
**   By setting the dos allocation strategy to LAST_FIT_LOW
**   before using DOS the set handle count function int 21h,
**   function 67h, DOS will allocate memory for the extended
**   file handle table at the end of free memory instead of
**   after the last heap allocation, with the benefit of
**   allowing the heap manager make further contiguous
**   allocations from the operating system.
**
*/

#include <stdlib.h>
#include <dos.h>
#include "strat.h"

/*
**   Gets dos memory allocation strategy via function 0x58.
**   Returns  allocation strategy, else returns -1 and sets
**   _doserrno on error.
*/

int get_alloc_strat(void)
{
      union REGS r;

      r.x.ax = 0x5800;              /* DOS "get allocation strategy" */
      int86(0x21,&r,&r);
      if (r.x.cflag)                /* error? */
      {
            _doserrno = r.x.ax;     /* save error code */
            return -1;
      }
      else  return r.x.ax;
}

/*
**   Sets DOS memory allocation strategy
**   returns allocation strategy on success,
**   else returns -1 and sets _doserrno on error
*/

int set_alloc_strat( int strat )
{
      union REGS r;

      r.x.ax = 0x5801;              /* DOS "set allocation strategy" */
      r.x.bx = strat;
      int86(0x21,&r,&r);
      if (r.x.cflag)                /* error? */
      {
            _doserrno = r.x.ax;     /* save error code */
            return -1;
      }
      _doserrno = 0;
      return strat;
}

/*
**   Uses dos function 67h to increase open file handle count.
**   Returns -1 and sets _doserrno on error, 0 otherwise.
*/

int set_handle_count( unsigned nhandles )
{
      union REGS r;

      r.x.ax = 0x6700;
      r.x.bx = nhandles;
      int86(0x21,&r,&r);
      if(r.x.cflag)
      {
            _doserrno = r.x.ax;
            return -1;
      }
      _doserrno = 0;
      return 0;
}

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

/*
**  returns maximum number of files that can be open
*/

int handle_count(void)
{
      int handles[500];
```

```
101 |       int i, result;
102 |
103 |       /* try allocating as many file handles as possible */
104 |
105 |       for (i = 0; i < 500; i++)
106 |       {
107 |             if( (handles[i]=open("NUL",O_WRONLY)) == -1 )
108 |                   break;
109 |       }
110 |       result = i;
111 |
112 |       /* close all files opened */
113 |
114 |       for (i--; i >= 0; i--)
115 |             close(handles[i]);
116 |       return result;
117 | }
118 |
119 |
120 | /*
121 | **   Memory test, returns number of kilobytes that
122 | **   can be allocated before failure.
123 | */
124 |
125 | int memtest(void)
126 | {
127 |       static void *mem[1024];
128 |       int i,result;
129 |
130 |       /* try allocating as many 1K blocks as possible */
131 |
132 |       for(i=0;i<1024;i++)
133 |       {
134 |             if( (mem[i]=malloc(1024)) == NULL )
135 |                   break;
136 |       }
137 |       result = i;                              /* save result */
138 |
139 |       /* free memory allocated */
140 |
141 |       for(i--; i >= 0; i--)
142 |             free(mem[i]);
143 |       return result;
144 | }
145 |
146 | #define checkdoserror(f) \
147 |    ((f==-1)?printf("%s failed, doserror = %#02x\n",#f,_doserrno):0)
148 |
149 | int main(void)
150 | {
151 |       int strat;
152 |
153 |       /* do pre-test diagnostics */
154 |
155 |       printf("allocated %d Kbytes before failure\n",memtest());
156 |       printf("opened %d files\n",handle_count());
157 |
158 |       strat = get_alloc_strat();  /* save current allocation strategy */
159 |       checkdoserror(set_alloc_strat(LAST_FIT_LOW));
160 |       puts("setting handle count to 50, with changed allocation strategy");
161 |       checkdoserror(set_handle_count(50));
162 |       checkdoserror(set_alloc_strat(strat)); /* restore strategy */
163 |
164 |       /* do post-test diagnostics */
165 |
166 |       printf("allocated %d Kbytes before failure\n",memtest());
167 |       printf("opened %d files\n",handle_count());
168 |
169 |       return 0;
170 | }
171 |
172 | #endif /* TEST */
```

TEXT STATISTICS

```
  4108 characters
   172 lines
```

LEXICAL STATISTICS

```
    23 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     8 constants [string]
    26 constants [numeric]
```

SYMBOL TABLE

```
    LAST_FIT_LOW           159
    NULL           134
    O_WRONLY       107
    REGS            32      52      73
    TEST            87
    _doserrno       38      59      62      80      83     147
    ax              34      38      41      54      59      75      80
    bx              55      76
    cflag           36      57      78
    checkdoserror          146     159     161     162
    close          115
    dos             21
    f      146     147+
    fcntl           92
    free           142
    get_alloc_strat         30     158
    h       20      21      89      90      91      92
    handle_count            98     156     167
    handles        100     107     115
    i      101     105+    107     110     114+    115     128     132+    134     137     141+    142
    int86           35      56      77
    io              91
    main           149
    malloc         134
    mem            127     134     142
    memtest        125     155     166
    nhandles        71      76
    open           107
    printf         147     155     156     166     167
    puts           160
    r       32      34      35+     36      38      41      52      54      55      56+     57      59
            73      75      76      77+     78      80
    result         101     110     116     128     137     143
    set_alloc_strat         50     159     162
    set_handle_count        71     161
    stdio           89
    stdlib          20      90
    strat           50      55      63     151     158     162
    x       34      36      38      41      54      55      57      59      75      76      78      80
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  strat.h 10-5-91  Robert Mashlan, public domain
  5 | **
  6 | **  header file for strat.c
  7 | **
  8 | */
  9 |
 10 | #ifndef STRAT__H
 11 | #define STRAT__H
 12 |
 13 | const enum {
 14 |       FIRST_FIT_LOW,
 15 |       BEST_FIT_LOW,
 16 |       LAST_FIT_LOW,
 17 |       FIRST_FIT_HIGH = 0x80, /* these strategies available only in DOS 5.0 */
 18 |       BEST_FIT_HIGH,
 19 |       LAST_FIT_HIGH,
 20 |       FIRST_FIT_HIGHONLY = 0x40,
 21 |       BEST_FIT_HIGHONLY,
 22 |       LAST_FIT_HIGHONLY
 23 |       };
 24 |
 25 | int get_alloc_strat(void);
 26 | int set_alloc_strat( int strat );
 27 | int set_handle_count( unsigned nhandles );
 28 |
 29 | #endif /* STRAT__H */
```

TEXT STATISTICS

```
   611 characters
    29 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    BEST_FIT_HIGH           18
    BEST_FIT_HIGHONLY       21
    BEST_FIT_LOW            15
    FIRST_FIT_HIGH          17
    FIRST_FIT_HIGHONLY              20
    FIRST_FIT_LOW           14
    LAST_FIT_HIGH           19
    LAST_FIT_HIGHONLY       22
    LAST_FIT_LOW            16
    STRAT__H        10      11
    get_alloc_strat         25
    nhandles        27
    set_alloc_strat         26
    set_handle_count        27
    strat           26
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)strcase
* @(#)  Converts a string to upper/lowercase using non ASCII convertion
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/
#include <string.h>                         /* strlen */
#include "bacstd.h"

/*
 /-------------------------------------\
 |          STRCASE                    |-----------------------------------|
 |\-----------------------------------/
 |
 | Converts a string to upper/lowercase using non ASCII convertion
 |
 | Examples of sort sequences found in SORTKEY.h
 |---------------------------------------------------------------------------|
 | CALL:
 |    pszStr = strcase("abc‘›†", (unsigned char *) CaseUpper) ;
 |
 | HEADER:
 |    sortkey.h
 |
 | GLOBALE VARIABLES:
 |    %
 |
 | ARGUMENTS:
 |    char *pszStr       Pointer to string
 |    char *pszOrder     Pointer to table with sort sequences
 |
 | PROTOTYPE:
 |    int _CfmTYPE strcase(unsigned char *pszStr, unsigned char *pszOrder) ;
 |
 | RETURN VALUE:
 |    char *        Pointer to string
 |
 | MODULE:
 |    STRCASE.c
 |---------------------------------------------------------------------------|
 |
 |---------------------------------------------------------------------------|
 |1993-04-10/Erik Bachmann
 \---------------------------------------------------------------------------|*/

unsigned char _CfnTYPE *strcase(unsigned char *pszStr, unsigned char *pszOrder)
{
    long        lCount  =   0L ;

    /*---------------------------------------------------------*/

    for ( lCount = 0L ; '\0' != pszStr[lCount] ; lCount++ )
    {
        pszStr[lCount] = pszOrder[ pszStr[lCount] ] ;
    }

    return( pszStr ) ;
}   /*** strcase() ***/

#ifdef  TEST

#include "sortkey.h"

int main()
{
    int i;
    char                *str    = "abcd‘›†„”„”,";

    /*---------------------------------------*/

    fprintf(stderr, "strcase()\n\n");

    printf("\nOriginal string : %s", str);

    strcase(str, (unsigned char *) CaseUpper);
    printf("\nUppercase string: %s", str);

    strcase(str, (unsigned char *) CaseLower);
    printf("\nLowercase string: %s", str);

    return( 0 ) ;
}   /*** main() ***/

#endif
```

TEXT STATISTICS

```
  3946 characters
    89 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
     7 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    CaseLower       83
    CaseUpper       80
    TEST            65
    _CfnTYPE        51
    fprintf         76
    h       12
    i       71
    lCount          53      57+     59+
    main            69
    printf          78      81      84
    pszOrder        51      59
    pszStr          51      57      59+     62
    stderr          76
    str             72      78      80      81      83      84
    strcase         51      80      83
    string          12
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STRCHCAT.C - Append a character to a string.
**
**  Arguments: 1 - Pointer to string to append to
**             2 - Character to append
**             3 - Maximum size of string buffer
**
**  Returns: Pointer to modified string, or NULL if insufficient space
**
**  Original Copyright 1990-95 by Robert B. Stout as part of
**  the MicroFirm Function Library (MFL)
**
**  The user is granted a free limited license to use this source file
**  to create royalty-free programs, subject to the terms of the
**  license restrictions specified in the LICENSE.MFL file.
**
**  NOTE: The name of this funtion violates ANSI/ISO 9899:1990 sec. 7.1.3,
**        but this violation seems preferable to either violating sec. 7.13.8
**        or coming up with some hideous mixed-case or underscore infested
**        naming. Also, many SNIPPETS str---() functions duplicate existing
**        functions which are supported by various vendors, so the naming
**        violation may be required for portability.
*/

#include <stdlib.h>
#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strchcat(char *string, int ch, size_t buflen)
{
      size_t len;

      if (NULL == string || ((len = strlen(string)) + 1) >= buflen)
            return NULL;

      string[len++] = ch;
      string[len]   = '\0';
      return string;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

main()
{
      char buf1[80] = "This buffer's big enough",
           buf2[] = "This one's not";
      char *ptr;

      printf("strchcat(\"%s\", '!') ", buf1);
      ptr = strchcat(buf1, '!', sizeof(buf1));
      printf("returned %p...", ptr);
      if (NULL != ptr)
            printf("\n...which is \"%s\"", buf1);
      puts("\n");

      printf("strchcat(\"%s\", '!') ", buf2);
      ptr = strchcat(buf2, '!', sizeof(buf2));
      printf("returned %p...", ptr);
      if (NULL != ptr)
            printf("\n...which is \"%s\"", buf2);
      puts("\n");
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2188 characters
    77 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     3 constants [character]
    12 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS            74
    NULL            39      40      64      71
    TEST            51
    __cplusplus             31+     47+
    buf1            57      61      62+     65
    buf2            58      68      69+     72
    buflen          35      39
    ch              35      42
    defined         31      47
    h       27      28      53
    len             37      39      42      43
    main            55
    printf          61      63      65      68      70      72
    ptr             59      62      63      64      69      70      71
    puts            66      73
    size_t          35      37
    stdio           53
    stdlib          27
    strchcat        35      62      69
    string          28      35      39+     42      43      44
    strlen          39
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STRDEL.C - Removes specified characters from a string
**
**  public domain demo by Bob Stout
**
**  NOTE: The name of this funtion violates ANSI/ISO 9899:1990 sec. 7.1.3,
**        but this violation seems preferable to either violating sec. 7.13.8
**        or coming up with some hideous mixed-case or underscore infested
**        naming. Also, many SNIPPETS str---() functions duplicate existing
**        functions which are supported by various vendors, so the naming
**        violation may be required for portability.
*/

#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strdel(char *str, size_t posn, size_t len)
{
      char *pos0, *pos1;

      if (str)
      {
            if (posn < strlen(str))
            {
                  for (pos0 = pos1 = str + posn;
                        *pos1 && len;
                        ++pos1, --len)
                  {
                        ;
                  }
                  strMove(pos0, pos1);
            }
      }
      return str;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      int pos, len;

      if (4 > argc)
      {
            puts("Usage: STRDEL string pos len");
            puts("Deletes 'len' characters starting at position 'pos'");
            return EXIT_FAILURE;
      }
      pos = atoi(argv[2]);
      len = atoi(argv[3]);
      printf("strdel(\"%s\", %d, %d) => ", argv[1], pos, len);
      printf("\"%s\"\n", strdel(argv[1], pos, len));
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1744 characters
    69 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     6 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            60
    EXIT_SUCCESS            66
    TEST            47
    __cplusplus             19+     43+
    argc            52      56
    argv            52      62      63      64      65
    atoi            62      63
    defined         19      43
    h       16      49      50
    len             23      32      33      54      63      64      65
    main            52
    pos             54      62      64      65
    pos0            25      31      37
    pos1            25      31      32      33      37
    posn            23      29      31
    printf          64      65
    puts            58      59
    size_t          23+
    stdio           49
    stdlib          50
    str             23      27      29      31      40
    strMove         37
    strdel          23      65
    string          16
    strlen          29
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STRDELCH.C - Removes specified character(s) from a string
**
**  public domain demo by Bob Stout
**
**  NOTE: The name of this funtion violates ANSI/ISO 9899:1990 sec. 7.1.3,
**        but this violation seems preferable to either violating sec. 7.13.8
**        or coming up with some hideous mixed-case or underscore infested
**        naming. Also, many SNIPPETS str---() functions duplicate existing
**        functions which are supported by various vendors, so the naming
**        violation may be required for portability.
*/

#include <stdio.h>
#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strdelch(char *string, const char *lose)
{
      if (!string || !*string)
            return NULL;
      if (lose)
      {
            char *s;

            for (s = string; *s; ++s)
            {
                  if (strchr(lose, *s))
                  {
                        strMove(s, s + 1);
                        --s;
                  }
            }
      }
      return string;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

main(int argc, char *argv[])
{
      char *strng, *delstrng;

      if (3 > argc--)
      {
            puts("Usage: STRDELCH char(s)_to_delete string_1 [...string_N]");
            return -1;
      }
      else  delstrng = *(++argv);
      while (--argc)
      {
            strng = *(++argv);
            printf("strdelch(\"%s\", \"%s\") => ", delstrng, strng);
            printf("\"%s\"\n", strdelch(strng, delstrng));
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1710 characters
    69 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     5 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    NULL            27
    TEST            48
    __cplusplus             20+     44+
    argc            50      54      60
    argv            50      59      62
    defined         20      44
    delstrng        52      59      63      64
    h       16      17
    lose            24      28      34
    main            50
    printf          63      64
    puts            56
    s       30      32+     34      36+     37
    stdio           16
    strMove         36
    strchr          34
    strdelch        24      64
    string          17      24      26+     32      41
    strng           52      62      63      64
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Portable, public domain strdup() by Bob Stout
  5 | */
  6 |
  7 | #include <stdlib.h>
  8 | #include <string.h>
  9 | #include "snip_str.h"
 10 |
 11 | #if defined(__cplusplus) && __cplusplus
 12 |  extern "C" {
 13 | #endif
 14 |
 15 | char *strdup(const char *string)
 16 | {
 17 |       char *newstring;
 18 |
 19 |       if (string)
 20 |       {
 21 |             if (NULL != (newstring = malloc(strlen(string) + 1)))
 22 |                   strcpy(newstring, string);
 23 |             return newstring;
 24 |       }
 25 |       else  return NULL;
 26 | }
 27 |
 28 | #if defined(__cplusplus) && __cplusplus
 29 |  }
 30 | #endif
```

TEXT STATISTICS

```
   566 characters
    30 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     2 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    NULL            21      25
    __cplusplus             11+     28+
    defined         11      28
    h       7        8
    malloc          21
    newstring       17      21      22      23
    stdlib           7
    strcpy          22
    strdup          15
    string           8      15      19      21      22
    strlen          21
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  STRECPY.C - A form of strcpy() where the result returned is the
  5 | **              NUL terminating character of the first argument. In many
  6 | **              cases this function is more efficient than the equivalent
  7 | **              strcpy, followed by strcat.
  8 | **
  9 | **  public domain by Leslie Satenstein and Thad Smith.
 10 | **
 11 | **  NOTE: The name of this funtion violates ANSI/ISO 9899:1990 sec. 7.1.3,
 12 | **        but this violation seems preferable to either violating sec. 7.13.8
 13 | **        or coming up with some hideous mixed-case or underscore infested
 14 | **        naming. Also, many SNIPPETS str---() functions duplicate existing
 15 | **        functions which are supported by various vendors, so the naming
 16 | **        violation may be required for portability.
 17 | */
 18 |
 19 | #include <stdio.h>
 20 | #include "sniptype.h"
 21 | #include "snip_str.h"
 22 |
 23 | #if defined(__cplusplus) && __cplusplus
 24 |  extern "C" {
 25 | #endif
 26 |
 27 | char *strecpy(char *target, const char *src)
 28 | {
 29 |       if (src && target)
 30 |       {
 31 |             while ((*target++ = *src++) != NUL)
 32 |                   ;
 33 |             return (--target);
 34 |       }
 35 |       else  return NULL;
 36 | }
 37 |
 38 | #if defined(__cplusplus) && __cplusplus
 39 |  }
 40 | #endif
```

TEXT STATISTICS

```
  1231 characters
    40 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     3 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    NUL             31
    NULL            35
    __cplusplus             23+     38+
    defined         23      38
    h       19
    src             27      29      31
    stdio           19
    strecpy         27
    target          27      29      31      33
```

```
/* +++Date last modified: 05-Jul-1997 */

/**
 *
 * strftime.c
 *
 * implements the ansi c function strftime()
 *
 * written 6 september 1989 by jim nutt
 * released into the public domain by jim nutt
 *
 * modified 21-Oct-89 by Rob Duff
 *
 **/

#include <stddef.h>     /* for size_t */
#include <stdarg.h>     /* for va_arg */
#include <time.h>       /* for struct tm */
#include "strftime.h"

static char *aday[] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};

static char *day[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
};

static char *amonth[] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};

static char *month[] = {
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
};

char *tzname_[2] = {"CST", "CDT"};        /* Add your own defaults here */

static char buf[26];

static void strfmt(char *str, const char *fmt, ...);

/**
 *
 * size_t strftime_(char *str,
 *                  size_t maxs,
 *                  const char *fmt,
 *                  const struct tm *t)
 *
 *      this functions acts much like a sprintf for time/date output.
 *      given a pointer to an output buffer, a format string and a
 *      time, it copies the time to the output buffer formatted in
 *      accordance with the format string.  the parameters are used
 *      as follows:
 *
 *          str is a pointer to the output buffer, there should
 *          be at least maxs characters available at the address
 *          pointed to by str.
 *
 *          maxs is the maximum number of characters to be copied
 *          into the output buffer, included the '\0' terminator
 *
 *          fmt is the format string.  a percent sign (%) is used
 *          to indicate that the following character is a special
 *          format character.  the following are valid format
 *          characters:
 *
 *              %A      full weekday name (Monday)
 *              %a      abbreviated weekday name (Mon)
 *              %B      full month name (January)
 *              %b      abbreviated month name (Jan)
 *              %c      standard date and time representation
 *              %d      day-of-month (01-31)
 *              %H      hour (24 hour clock) (00-23)
 *              %I      hour (12 hour clock) (01-12)
 *              %j      day-of-year (001-366)
 *              %M      minute (00-59)
 *              %m      month (01-12)
 *              %p      local equivalent of AM or PM
 *              %S      second (00-59)
 *              %U      week-of-year, first day sunday (00-53)
 *              %W      week-of-year, first day monday (00-53)
 *              %w      weekday (0-6, sunday is 0)
 *              %X      standard time representation
 *              %x      standard date representation
 *              %Y      year with century
 *              %y      year without century (00-99)
 *              %Z      timezone name
 *              %%      percent sign
 *
 *      the standard date string is equivalent to:
 *
 *          %a %b %d %Y
 *
 *      the standard time string is equivalent to:
 *
 *          %H:%M:%S
```

```
 *
 *      the standard date and time string is equivalent to:
 *
 *          %a %b %d %H:%M:%S %Y
 *
 *      strftime_() returns the number of characters placed in the
 *      buffer, not including the terminating \0, or zero if more
 *      than maxs characters were produced.
 *
 **/

size_t strftime_(char *s, size_t maxs, const char *f, const struct tm *t)
{
      int w;
      char *p, *q, *r;

      p = s;
      q = s + maxs - 1;
      while ((*f != '\0'))
      {
            if (*f++ == '%')
            {
                  r = buf;
                  switch (*f++)
                  {
                  case '%' :
                        r = "%";
                        break;

                  case 'a' :
                        r = aday[t->tm_wday];
                        break;

                  case 'A' :
                        r = day[t->tm_wday];
                        break;

                  case 'b' :
                        r = amonth[t->tm_mon];
                        break;

                  case 'B' :
                        r = month[t->tm_mon];
                        break;

                  case 'c' :
                        strfmt(r, "%0 %0 %2 %2:%2:%2 %4",
                              aday[t->tm_wday], amonth[t->tm_mon],
                              t->tm_mday,t->tm_hour, t->tm_min,
                              t->tm_sec, t->tm_year+1900);
                        break;

                  case 'd' :
                        strfmt(r,"%2",t->tm_mday);
                        break;

                  case 'H' :
                        strfmt(r,"%2",t->tm_hour);
                        break;

                  case 'I' :
                        strfmt(r,"%2",(t->tm_hour%12)?t->tm_hour%12:12);
                        break;

                  case 'j' :
                        strfmt(r,"%3",t->tm_yday+1);
                        break;

                  case 'm' :
                        strfmt(r,"%2",t->tm_mon+1);
                        break;

                  case 'M' :
                        strfmt(r,"%2",t->tm_min);
                        break;

                  case 'p' :
                        r = (t->tm_hour>11)?"PM":"AM";
                        break;

                  case 'S' :
                        strfmt(r,"%2",t->tm_sec);
                        break;

                  case 'U' :
                        w = t->tm_yday/7;
                        if (t->tm_yday%7 > t->tm_wday)
                              w++;
                        strfmt(r, "%2", w);
                        break;

                  case 'W' :
                        w = t->tm_yday/7;
                        if (t->tm_yday%7 > (t->tm_wday+6)%7)
                              w++;
                        strfmt(r, "%2", w);
                        break;

                  case 'w' :
                        strfmt(r,"%1",t->tm_wday);
```

```
201 |                         break;
202 |
203 |                   case 'x' :
204 |                         strfmt(r, "%3s %3s %2 %4", aday[t->tm_wday],
205 |                               amonth[t->tm_mon], t->tm_mday, t->tm_year+1900);
206 |                         break;
207 |
208 |                   case 'X' :
209 |                         strfmt(r, "%2:%2:%2", t->tm_hour,
210 |                               t->tm_min, t->tm_sec);
211 |                         break;
212 |
213 |                   case 'y' :
214 |                         strfmt(r,"%2",t->tm_year%100);
215 |                         break;
216 |
217 |                   case 'Y' :
218 |                         strfmt(r,"%4",t->tm_year+1900);
219 |                         break;
220 |
221 |                   case 'Z' :
222 |                         r = (t->tm_isdst && tzname_[1][0]) ?
223 |                               tzname_[1] : tzname_[0];
224 |                         break;
225 |
226 |                   default:
227 |                         buf[0] = '%';     /* reconstruct the format */
228 |                         buf[1] = f[-1];
229 |                         buf[2] = '\0';
230 |                         if (buf[1] == 0)
231 |                               f--;        /* back up if at end of string */
232 |                   }
233 |                   while (*r)
234 |                   {
235 |                         if (p == q)
236 |                         {
237 |                               *q = '\0';
238 |                               return 0;
239 |                         }
240 |                         *p++ = *r++;
241 |                   }
242 |             }
243 |             else
244 |             {
245 |                   if (p == q)
246 |                   {
247 |                         *q = '\0';
248 |                         return 0;
249 |                   }
250 |                   *p++ = f[-1];
251 |             }
252 |       }
253 |       *p = '\0';
254 |       return p - s;
255 | }
256 |
257 | /*
258 |  *  stdarg.h
259 |  *
260 | typedef void *va_list;
261 | #define va_start(vp,v) (vp=((char*)&v)+sizeof(v))
262 | #define va_arg(vp,t) (*((t*)(vp))++)
263 | #define va_end(vp)
264 |  *
265 |  */
266 |
267 | static int pow[5] = { 1, 10, 100, 1000, 10000 };
268 |
269 | /**
270 |  * static void strfmt(char *str, char *fmt);
271 |  *
272 |  * simple sprintf for strftime
273 |  *
274 |  * each format descriptor is of the form %n
275 |  * where n goes from zero to four
276 |  *
277 |  * 0    -- string %s
278 |  * 1..4 -- int %?.?d
279 |  *
280 | **/
281 |
282 | static void strfmt(char *str, const char *fmt, ...)
283 | {
284 |       int ival, ilen;
285 |       char *sval;
286 |       va_list vp;
287 |
288 |       va_start(vp, fmt);
289 |       while (*fmt)
290 |       {
291 |             if (*fmt++ == '%')
292 |             {
293 |                   ilen = *fmt++ - '0';
294 |                   if (ilen == 0)                /* zero means string arg */
295 |                   {
296 |                         sval = va_arg(vp, char*);
297 |                         while (*sval)
298 |                               *str++ = *sval++;
299 |                   }
300 |                   else                          /* always leading zeros */
```

```
301 |                   {
302 |                         ival = va_arg(vp, int);
303 |                         while (ilen)
304 |                         {
305 |                               ival %= pow[ilen--];
306 |                               *str++ = (char)('0' + ival / pow[ilen]);
307 |                         }
308 |                   }
309 |             }
310 |             else  *str++ = fmt[-1];
311 |       }
312 |       *str = '\0';
313 |       va_end(vp);
314 | }
315 |
316 | #ifdef TEST
317 |
318 | #include <stdio.h>      /* for printf */
319 | #include <time.h>       /* for strftime */
320 |
321 | char test[80];
322 |
323 | int main(int argc, char *argv[])
324 | {
325 |       int len;
326 |       char *fmt;
327 |       time_t now;
328 |
329 |       time(&now);
330 |
331 |       fmt = (argc == 1) ? "%I:%M %p\n%c\n" : argv[1];
332 |       len = strftime(test,sizeof test, fmt, localtime(&now));
333 |       printf("%d: %s\n", len, test);
334 |       return !len;
335 | }
336 |
337 | #endif /* TEST */
```

TEXT STATISTICS

```
  9974 characters
   337 lines
```

LEXICAL STATISTICS

```
    16 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
    33 constants [character]
    61 constants [string]
    43 constants [numeric]
```

SYMBOL TABLE

```
    TEST            316
    aday             21     131     148     204
    amonth           30     139     148     205
    argc            323     331
    argv            323     331
    buf              42     123     227     228     229     230
    day              25     135
    f       112     119     121     124     228     231     250
    fmt              44     282     288     289     291     293     310     326     331     332
    h        16      17      18     318     319
    ilen            284     293     294     303     305     306
    ival            284     302     305     306
    len             325     332     333     334
    localtime       332
    main            323
    maxs            112     118
    month            35     143
    now             327     329     332
    p       115     117     235     240     245     250     253     254
    pow             267     305     306
    printf          333
    q       115     118     235     237     245     247
    r       115     123     127     131     135     139     143     147     154     158     162     166
            170     174     178     182     189     196     200     204     209     214     218     222
            233     240
    s       112     117     118     254
    size_t          112+
    stdarg           17
    stddef           16
    stdio           318
    str              44     282     298     306     310     312
    strfmt           44     147     154     158     162     166     170     174     182     189     196
            200     204     209     214     218     282
    strftime        332
    strftime_       112
    sval            285     296     297     298
    t       112     131     135     139     143     148+    149+    150+    154     158     162+    166
            170     174     178     182     186     187+    193     194+    200     204     205+    209
            210+    214     218     222
    test            321     332+    333
    time             18     319     329
    time_t          327
    tm              112
    tm_hour         149     158     162+    178     209
    tm_isdst        222
    tm_mday         149     154     205
    tm_min          149     174     210
    tm_mon          139     143     148     170     205
    tm_sec          150     182     210
    tm_wday         131     135     148     187     194     200     204
    tm_yday         166     186     187     193     194
    tm_year         150     205     214     218
    tzname_          40     222     223+
    va_arg          296     302
    va_end          313
    va_list         286
    va_start        288
    vp              286     288     296     302     313
    w       114     186     188     189     193     195     196
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  STRFTIME.H - For older compilers which lack strftime()
   5 | **
   6 | **  Note: To avoid name collision with newer compilers, the function name
   7 | **      strftime_() is used.
   8 | */
   9 |
  10 | #ifndef STRFTIME__H
  11 | #define STRFTIME__H
  12 |
  13 | #include <stddef.h>     /* for size_t */
  14 | #include <time.h>       /* for struct tm */
  15 |
  16 | size_t strftime_(char *s, size_t maxs, const char *f, const struct tm *t);
  17 |
  18 | extern char * tzname_[2];
  19 |
  20 | #endif /* STRFTIME__H */
```

TEXT STATISTICS

```
   488 characters
    20 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    STRFTIME__H             10      11
    f       16
    h       13      14
    maxs            16
    s       16
    size_t          16+
    stddef          13
    strftime_       16
    t       16
    time            14
    tm              16
    tzname_         18
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  STRINGIZ.H - Macros to use the ANSI/ISO preprocessor "stringize" macro
  5 | */
  6 |
  7 | #ifndef STRINGIZ__H
  8 | #define STRINGIZ__H
  9 |
 10 | #define stringize(y) #y
 11 | #define add_quotes(x) stringize (x)
 12 |
 13 | #ifdef TEST
 14 |
 15 | #include <stdio.h>
 16 | #include <string.h>
 17 |
 18 | #define FILENAME  MYFILE.XYZ
 19 |
 20 | int main()
 21 | {
 22 |       char tmp[80];
 23 |
 24 |       strcpy(tmp, "File: ");
 25 |       strcat(tmp, add_quotes(FILENAME));
 26 |       printf("%s\n", tmp);
 27 |       return 0;
 28 | }
 29 |
 30 | #endif /* TEST */
 31 |
 32 | #endif /* STRINGIZ__H */
```

TEXT STATISTICS

```
   535 characters
    32 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     2 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    FILENAME        18      25
    MYFILE          18
    STRINGIZ__H              7       8
    TEST            13
    XYZ             18
    add_quotes              11      25
    h       15      16
    main            20
    printf          26
    stdio           15
    strcat          25
    strcpy          24
    string          16
    stringize       10      11
    tmp             22      24      25      26
    x       11+
    y       10+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STRIPEOF.C
**
**  public domain demo by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

#define BUFSIZE 16384

int main(int argc, char *argv[])
{
      char *buf;

      if (2 > argc)
      {
            puts("Usage: STRIPEOF filename1 [...filenameN]");
            return EXIT_FAILURE;
      }
      if (NULL == (buf = malloc(BUFSIZE)))
      {
            puts("STRIPEOF internal failure");
            return EXIT_FAILURE;
      }
      while (--argc)
      {
            int fd;
            size_t bytes;
            int found = 0;
            long zpos = 0L;

            if (-1 == (fd = open(*(++argv), O_RDWR | O_BINARY)))
            {
                  printf("Couldn't open %s\n", *argv);
                  return EXIT_FAILURE;
            }
            while (0 < (bytes = read(fd, buf, BUFSIZE)))
            {
                  int i;

                  for (i = 0; i < (int)bytes; ++i)
                  {
                        if (('Z' - 64) == buf[i])
                        {
                              found = 1;
                              zpos += i;
                              break;
                        }
                  }
                  if (found)
                        break;
                  zpos += bytes;
            }
            if (found)
                  chsize(fd, zpos);
      }
      return EXIT_SUCCESS;
}
```

```
TEXT STATISTICS

  1517 characters
    63 lines

LEXICAL STATISTICS

     2 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
     3 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    BUFSIZE          14      25      42
    EXIT_FAILURE             23      28      40
    EXIT_SUCCESS             62
    NULL             25
    O_BINARY         37
    O_RDWR           37
    argc             16      20      30
    argv             16      37      39
    buf              18      25      42      48
    bytes            33      42      46      57
    chsize           60
    fcntl            12
    fd               32      37      42      60
    found            34      50      55      59
    h         9      10      11      12
    i        44      46+     48      51
    io               11
    main             16
    malloc           25
    open             37
    printf           39
    puts             22      27
    read             42
    size_t           33
    stdio             9
    stdlib           10
    zpos             35      51      57      60
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Designation:  StriStr
**
** Call syntax:  char *stristr(char *String, char *Pattern)
**
** Description:  This function is an ANSI version of strstr() with
**               case insensitivity.
**
** Return item:  char *pointer if Pattern is found in String, else
**               pointer to 0
**
** Rev History:  07/04/95  Bob Stout  ANSI-fy
**               02/03/94  Fred Cole  Original
**
** Hereby donated to public domain.
*/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "snip_str.h"

typedef unsigned int uint;

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *stristr(const char *String, const char *Pattern)
{
      char *pptr, *sptr, *start;
      uint  slen, plen;

      for (start = (char *)String,
           pptr  = (char *)Pattern,
           slen  = strlen(String),
           plen  = strlen(Pattern);

           /* while string length not shorter than pattern length */

           slen >= plen;

           start++, slen--)
      {
            /* find start of pattern in string */
            while (toupper(*start) != toupper(*Pattern))
            {
                  start++;
                  slen--;

                  /* if pattern longer than string */

                  if (slen < plen)
                        return(NULL);
            }

            sptr = start;
            pptr = (char *)Pattern;

            while (toupper(*sptr) == toupper(*pptr))
            {
                  sptr++;
                  pptr++;

                  /* if end of pattern then pattern was found */

                  if ('\0' == *pptr)
                        return (start);
            }
      }
      return(NULL);
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

int main(void)
{
      char buffer[80] = "heLLo, HELLO, hello, hELLo, HellO";
      char *sptr = buffer;

      while (0 != (sptr = stristr(sptr, "hello")))
            printf("Found %5.5s!\n", sptr++);

      return(0);
}

#endif /* TEST */
```

TEXT STATISTICS

```
  2128 characters
    93 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     1 constants [character]
     5 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    NULL            56      73
    Pattern         31      37      39      48      60
    String          31      36      38
    TEST            80
    __cplusplus             27+     76+
    buffer          84      85
    ctype           22
    defined         27      76
    h       20      21      22
    main            82
    plen            34      39      43      55
    pptr            33      37      60      62      65      69
    printf          88
    slen            34      38      43      45      51      55
    sptr            33      59      62      64      85      87+     88
    start           33      36      45      48      50      59      70
    stdio           20
    string          21
    stristr         31      87
    strlen          38      39
    toupper         48+     62+
    uint            25      34
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)strnsub
* @(#)-     searches string for a given pattern and replaces it with
* @(#)      a new text string.
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include "bacstd.h"
#include <stdio.h>
#include <string.h>

/*
 /-------------------------------------\
|        STRNSUB                       |-----------------------------------|
|\-------------------------------------/
|    Function searches string for a given pattern and replaces it with
| a new text string.
|
| Note that the pattern string is replaced with the
| replacement string and they do not have to be the same length. It is
| assumed, however, that the original string is long enough to hold the
| result. Maximum length + 1 is given in iMaxLength
|
|
|---------------------------------------------------------------------------|
| CALL:
|    strnsub(pszStreng, org, rep, maxlength) ;
|
| HEADER:
|    bacstd.h
|    stdio.h
|    string.h
|    malloc.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    char *pszString    the string to search
|    char *pat          the pattern to locate
|    char *rep          the replacement string
|    int  iMaxLength    the maximum length of string after replacement
|
| PROTOTYPE:
|    char CfnTYPE *strnsub(char *pszString, char *pat, char *rep,
|                          int iMaxLength) ;
|
| RETURN VALUE:
|    char *         pointer to the replaced string
|
| MODULE:
|    strsub.c
|---------------------------------------------------------------------------|
|
|
|
|---------------------------------------------------------------------------|
|1993-03-25/Erik Bachmann
\---------------------------------------------------------------------------|*/

char _CfnTYPE *strnsub(char *pszString,
                       char *pszPattern,
                       char *pszReplacement,
                       int iMaxLength)
{
      char  *pszSubstring,
                  *pszTmpSubstring ;

      int   iPatternLength,
                  iReplacementLength ;

      /*-------------------------------------------------*/

      pszTmpSubstring = pszSubstring = pszString ;
      iPatternLength = strlen(pszPattern) ;
      iReplacementLength = strlen(pszReplacement) ;

      if ( 0 == strcmp( pszPattern, pszReplacement ) )
            return( 0 ) ;                 /* Pattern == replacement: loop */

      if ( NULL == ( pszSubstring = strstr(pszString, pszPattern ) ) )
      {                                   /* No match found               */
            return( NULL );
      }

      if ( ( strlen( pszString ) + ( iReplacementLength - iPatternLength ) )
             >= iMaxLength )

            /* Enough space for replacement? */

            return( NULL ) ;

      pszTmpSubstring = (char *) calloc(iMaxLength, sizeof(char)) ;

```

```
101 |       /* Buy some workspace      */
102 |
103 |       if ( NULL == pszTmpSubstring )            /* No memory left          */
104 |             return( NULL ) ;
105 |
106 |       strcpy( pszTmpSubstring, pszSubstring + iPatternLength ) ;
107 |
108 |       /* If there was space left */
109 |
110 |       while ( iReplacementLength-- )
111 |       {     /* Copy replacement        */
112 |             *pszSubstring++ = *pszReplacement++ ;
113 |       }
114 |
115 |       strcpy( pszSubstring, pszTmpSubstring ) ;
116 |
117 |       /* Add old stuff back      */
118 |
119 |       free( pszTmpSubstring ) ;
120 |
121 |       return( pszSubstring - iPatternLength ) ;
122 |
123 |       /* Return pointer to change*/
124 | }
125 |
126 | #ifdef      TEST
127 |
128 | int main()
129 | {
130 |       char  szStreng[20] ;
131 |       char  *org  = "xx" ;
132 |       char  *rep  = "YYYY" ;
133 |
134 |       /*-------------------------------------------------*/
135 |
136 |       fprintf(stderr, "strnsub()\n\n") ;
137 |
138 |       strcpy(szStreng, "testxxtest") ;
139 |
140 |       fprintf(stderr, "Replacing %s with %s\n\nBefore : %s",
141 |               org, rep, szStreng) ;
142 |
143 |       if ( NULL == strnsub(szStreng, org, rep, 20) )
144 |             fprintf(stderr, "\nERROR\n\a") ;
145 |       else
146 |             fprintf(stderr, "\nOK\n") ;
147 |
148 |       fprintf(stderr, "After : %s\n\n", szStreng) ;
149 |
150 | }
151 |
152 | #endif
```

TEXT STATISTICS

```
  4365 characters
   152 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     9 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    NULL            87      89      97     103     104     143
    TEST           126
    _CfnTYPE        67
    calloc          99
    fprintf        136     140     144     146     148
    free           119
    h       15      16
    iMaxLength              70      93      99
    iPatternLength          75      81      92     106     121
    iReplacementLength              76      82      92     110
    main           128
    org            131     141     143
    pszPattern              68      81      84      87
    pszReplacement          69      82      84     112
    pszString       67      80      87      92
    pszSubstring            72      80      87     106     112     115     121
    pszTmpSubstring         73      80      99     103     106     115     119
    rep            132     141     143
    stderr         136     140     144     146     148
    stdio           15
    strcmp          84
    strcpy         106     115     138
    string          16
    strlen          81      82      92
    strnsub         67     143
    strstr          87
    szStreng       130     138     141     143     148
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)strrepc
* @(#)      Replaces all occurrences of a character in a string with another
* @(#)      character
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <string.h>
#include "bacstd.h"

/*
 /-------------------------------------\
|       STRREPC                        |-----------------------------------|
|\-------------------------------------/
|
| Replaces all occurrences of a character in a string with another character
|
|-----------------------------------------------------------------------------|
| CALL:
|    strrepc( str, cFrom, cTo ) ;
|
| HEADER:
|    string.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|    cFrom       : Char to be replaced
|    cTo         : Replacement
|
| PROTOTYPE:
|    int _CfnTYPE strrepc( char *pszStr, char cFrom, char cTo ) ;
|
| RETURN VALUE:
|    iReturn     : No of replacements
|
| MODULE:
|    strrepc.c
|-----------------------------------------------------------------------------|
|
|
|-----------------------------------------------------------------------------|
|1992-11-09/Erik Bachmann
\-----------------------------------------------------------------------------|*/

int _CfnTYPE strrepc(char *pszStr, char cFrom, char cTo)
{
      char  *ptr ;                          /* Pointer to string */
      int   iReturn = 0 ;                   /* No of replacements */

      /*-------------------------------------------------------------*/

      while( 0 != ( ptr = strchr( pszStr, cFrom ) ) )

      {     /* WHILE cFrom occurs in pszStr */

            pszStr[ (int) ptr - (int) pszStr ] = cTo ;

            /*- Replace next cFrom with cTo */

            iReturn++ ;                                         /*- count */
      }

      return( iReturn ) ;
}

#ifdef      TEST

int main()
{
      char streng[20];

      /*---------------------------------------*/

      strcpy(streng, "abcabcabc");

      fprintf(stderr, "strrepc()\n\n");

      fprintf(stderr, "Replacing c with C\n\nBefore : %s\n\n", streng);

      strrepc(streng, 'c', 'C');

      fprintf(stderr, "After  : %s\n\n", streng);

      return( 0 ) ;
}

#endif
```

```
TEXT STATISTICS

  2549 characters
    96 lines

LEXICAL STATISTICS

    10 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     2 constants [character]
     5 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    TEST            75
    _CfnTYPE        54
    cFrom           54      61
    cTo             54      65
    fprintf         85      87      91
    h       14
    iReturn         57      69      72
    main            77
    pszStr          54      61      65+
    ptr             56      61      65
    stderr          85      87      91
    strchr          61
    strcpy          83
    streng          79      83      87      89      91
    string          14
    strrepc         54      89
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
   ------------------------------------------------------------------
   Module:     REPLACE.C
   Author:     Gilles Kohl
   Started:    09.06.1992   12:16:47
   Modified:   09.06.1992   12:41:41
               22-Sep-95    Bob Stout
   Subject:    Replace one string by another in a given buffer.
               This code is public domain. Use freely.
   ------------------------------------------------------------------
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "snip_str.h"

/*
** strrepl: Replace OldStr by NewStr in string Str contained in buffer
**          of size BufSiz.
**
** Str should have enough allocated space for the replacement - if not,
** NULL is returned. Str and OldStr/NewStr should not overlap.
**
** The empty string ("") is found at the beginning of every string.
**
** Returns: pointer to first location behind where NewStr was inserted.
**          Str if OldStr was not found.
**          NULL if replacement would overflow Str's buffer
**
** This is useful for multiple replacements, see example in main() below
** (be careful not to replace the empty string this way !)
**
**  NOTE: The name of this funtion violates ANSI/ISO 9899:1990 sec. 7.1.3,
**        but this violation seems preferable to either violating sec. 7.13.8
**        or coming up with some hideous mixed-case or underscore infested
**        naming. Also, many SNIPPETS str---() functions duplicate existing
**        functions which are supported by various vendors, so the naming
**        violation may be required for portability.
*/

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strrepl(char *Str, size_t BufSiz, char *OldStr, char *NewStr)
{
      int OldLen, NewLen;
      char *p, *q;

      if(NULL == (p = strstr(Str, OldStr)))
            return Str;
      OldLen = strlen(OldStr);
      NewLen = strlen(NewStr);
      if ((strlen(Str) + NewLen - OldLen + 1) > BufSiz)
            return NULL;
      memmove(q = p+NewLen, p+OldLen, strlen(p+OldLen)+1);
      memcpy(p, NewStr, NewLen);
      return q;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

/*
** Test main().
** Given two arguments, replaces the first arg. in the lines read from
** stdin by the second one.
** Example invocation:
** replace printf puts <replace.c
** will replace all printf's by puts in replace's source.
**
*/

int main(int argc, char *argv[])
{
      char buf[200];
      char *Start, *Str;
      size_t BufSiz, BufLeft;

      if(argc < 3)
      {
            puts("Usage: STRREPL old_string new_string [buffer_size]");
            return EXIT_FAILURE;
      }
      if (argc > 3)
      {
            BufSiz = atoi(argv[3]);
            if (200 < BufSiz)
                  BufSiz = 200;
      }
      else  BufSiz = 20;            /* Pretend we have a short buffer   */

      /* Repeat until all occurrences replaced */

```

```
 101 |       while(NULL != (Start = fgets(buf, 200, stdin)))
 102 |       {
 103 |             if ('\n' == LAST_CHAR(Start))
 104 |                   LAST_CHAR(Start) = NUL;       /* Strip trailing \n    */
 105 |             for (Str = Start, BufLeft = BufSiz; Start != NULL; )
 106 |             {
 107 |                   Start = strrepl(Start, BufLeft, argv[1], argv[2]);
 108 |                   if (Start == Str)
 109 |                         break;
 110 |                   BufLeft -= Start - Str;
 111 |                   Str = Start;
 112 |             }
 113 |
 114 |             if (NULL == Start)
 115 |                   puts("\a*** Modified string will not fit in buffer ***");
 116 |             else  printf("%s\n", buf);
 117 |       }
 118 |       return EXIT_SUCCESS;
 119 | }
 120 |
 121 | #endif /* TEST */
 122 |
```

TEXT STATISTICS

```
  3760 characters
   122 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     1 constants [character]
     5 constants [string]
    12 constants [numeric]
```

SYMBOL TABLE

```
    BufLeft          84     105     107     110
    BufSiz           48      57      84      93      94      95      97     105
    EXIT_FAILURE             89
    EXIT_SUCCESS            118
    LAST_CHAR       103     104
    NUL             104
    NULL             53      58     101     105     114
    NewLen           50      56      57      59      60
    NewStr           48      56      60
    OldLen           50      55      57      59+
    OldStr           48      53      55
    Start            83     101     103     104     105+    107+    108     110     111     114
    Str              48      53      54      57      83     105     108     110     111
    TEST             68
    __cplusplus              44+     64+
    argc             80      86      91
    argv             80      93     107+
    atoi             93
    buf              82     101     116
    defined          44      64
    fgets           101
    h       15       16      17
    main             80
    memcpy           60
    memmove          59
    p       51       53      59+     60
    printf          116
    puts             88     115
    q       51       59      61
    size_t           48      84
    stdin           101
    stdio            15
    stdlib           16
    string           17
    strlen           55      56      57      59
    strrepl          48     107
    strstr           53
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STRREV.C - reverse a string in place
**
**  public domain by Bob Stout
*/

#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strrev(char *str)
{
      char *p1, *p2;

      if (! str || ! *str)
            return str;
      for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
      {
            *p1 ^= *p2;
            *p2 ^= *p1;
            *p1 ^= *p2;
      }
      return str;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>

int main(int argc, char *argv[])
{
      while (--argc)
      {
            printf("\"%s\" backwards is ", *++argv);
            printf("\"%s\"\n", strrev(*argv));
      }
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

   849 characters
    49 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     4 constants [string]
     2 constants [numeric]

SYMBOL TABLE

    TEST            35
    __cplusplus             12+     31+
    argc            39      41
    argv            39      43      44
    defined         12      31
    h       9       37
    main            39
    p1              18      22+     24      25      26
    p2              18      22+     24      25      26
    printf          43      44
    stdio           37
    str             16      20+     21      22+     28
    string           9
    strlen          22
    strrev          16      44
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** strrpbrk() - reverse of strpbrk() - Finds the last occurrence of
**              any characters from szChars found in szString.
**
** Donated to SNIPPETS by Phi Nguyen 1995, modified by Bob Stout
*/

#include <stdio.h>
#include <string.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strrpbrk(const char *szString, const char *szChars)
{
      const char  *p;
      char        *p0, *p1;

      for (p = szChars, p0 = p1 = NULL; p && *p; ++p)
      {
            p1 = strrchr(szString, *p);
            if (p1 && p1 > p0)
                  p0 = p1;
      }
      return p0;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

main()
{
      char  string[] = "This is a testing string",
            chars[]  = "xyzet",
           *ptr;

      ptr = strrpbrk(string, chars);

      if (ptr)
            printf("One or more of \"%s\" found at \"%s\"\n", chars, ptr);
      else  printf("Can't find any of \"%s\" in \"%s\".\n", chars, string);
      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1138 characters
    52 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     6 constants [string]
     1 constants [numeric]

SYMBOL TABLE

    NULL            23
    TEST            36
    __cplusplus             14+     32+
    chars           41      44      47      48
    defined         14      32
    h       10      11
    main            38
    p       20      23+     25
    p0              21      23      26      27      29
    p1              21      23      25      26+     27
    printf          47      48
    ptr             42      44      46      47
    stdio           10
    string          11      40      44      48
    strrchr         25
    strrpbrk        18      44
    szChars         18      23
    szString        18      25
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  strsort() -- Shell sort an array of string pointers via strcmp()
 5 | **  Written in ANSI C and optimized for size under Borland TC and BC++.
 6 | **
 7 | **  Public domain by Raymond Gardner 12/05/91   :-)
 8 | **  based on a public domain version by Thad Smith 12/05/91,
 9 | **  based on a public domain version by
10 | **  Ray Gardner   Denver, CO   12/88
11 | */
12 |
13 | #include <string.h>
14 | #include "snipsort.h"
15 |
16 | void STRSORT(char **v, unsigned n)
17 | {
18 |       register unsigned int gap;
19 |       unsigned int i, j;
20 |       register char **a;
21 |       char **b;
22 |
23 |       gap = 0;
24 |       do
25 |       {
26 |             gap <<= 1;
27 |       } while (++gap < n);
28 |
29 |       while ((i = (gap >>= 1)) != 0)
30 |       {
31 |             for ( ; (j = i) < n; i++)
32 |             {
33 |                   a = v + j;
34 |                   do
35 |                   {
36 |                         j -= gap;
37 |                         b = a;
38 |                         a -= gap;
39 |                         if (strcmp(*a, *b) > 0)
40 |                         {
41 |                               register char *tmp;
42 |                               tmp = *a;
43 |                               *a = *b;
44 |                               *b = tmp;
45 |                         }
46 |                         else  break;      /* better have this break! */
47 |                   } while (j >=  gap);
48 |             }
49 |       }
50 | }
```

```
TEXT STATISTICS

  1366 characters
    50 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     0 constants [character]
     1 constants [string]
     5 constants [numeric]

SYMBOL TABLE

    STRSORT         16
    a       20      33      37      38      39      42      43
    b       21      37      39      43      44
    gap             18      23      26      27      29      36      38      47
    h       13
    i       19      29      31+
    j       19      31      33      36      47
    n       16      27      31
    strcmp          39
    string          13
    tmp             41      42      44
    v       16      33
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)strtrim.c
* @(#)strtrimr - Removes all trailing blanks from a string.
* @(#)strtriml - Removes all leading blanks from a string.
* @(#)strtrim  - Removes all leading and trailing blanks in a string.
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <string.h>
#include <ctype.h>
#include "bacstd.h"

/*
 /-------------------------------------\
|  STRTRIMR                            |-------------------------------------|
|\-------------------------------------/
|
| Removes all trailing blanks from a string.
| Blanks are defined with ISSPACE  (blank, tab, newline, return, formfeed,
| vertical tab = 0x09 - 0x0D + 0x20)
|
|-----------------------------------------------------------------------------|
| CALL:
|    strtrimr(&str);
|
| HEADER:
|    ctype.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|
| PROTOTYPE:
|    int _CfnTYPE strtrimr(char *pszStr);
|
| RETURN VALUE:
|    j-i         : No of removed blanks
|
| MODULE:
|    strtrim.c
|-----------------------------------------------------------------------------|
|1994-01-08/Bac
|   All characters is checked (">" -> ">=").
|
|
|
|-----------------------------------------------------------------------------|
|1992-11-09/Erik Bachmann
\-----------------------------------------------------------------------------|*/
int _CfnTYPE strtrimr(char *pszStr)
{
      int   i, j;                               /* Local counters */

      /*-------------------------------------------------*/

      j = i = strlen(pszStr) - 1; /* Calculate the length of the string */

      while (isspace(pszStr[i]) && (i >= 0))

            /* WHILE string ends with a blank */
            /*1994-01-08/Bac Even if all chars are blanks (= 0) */

            pszStr[ i-- ] = '\0';               /*- Replace blank with '\0' */

      return(j - i);                            /* Return no of replacements */
}

/*
 /-------------------------------------\
|  STRTRIML                            |-------------------------------------|
|\-------------------------------------/
|
| Removes all leading blanks from a string.
| Blanks are defined with ISSPACE  (blank, tab, newline, return, formfeed,
| vertical tab = 0x09 - 0x0D + 0x20)
|
|
|-----------------------------------------------------------------------------|
| CALL:
|    strtriml(&str);
|
| HEADER:
|    ctype.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|
| PROTOTYPE:
|    int _CfnTYPE strtriml(char *pszStr);
```

```
|
| RETURN VALUE:
|    i           : No of removed blanks
|
| MODULE:
|    strtrim.c
|-------------------------------------------------------------------------|
|
|
|-------------------------------------------------------------------------|
|1992-11-09/Erik Bachmann
\-------------------------------------------------------------------------|*/

int _CfnTYPE strtriml(char *pszStr)
{
      int   i = 0, j;                              /* Local counters */

      /*-------------------------------------------------*/

      j = strlen(pszStr) - 1; /* Calculate the length of the string */

      while (isspace(pszStr[i]) && (i <= j))

            /* WHILE string starts with a blank */

            i++;                          /*- Count no of leading blanks */

      if (0 < i)                          /* IF leading blanks are found */
            strcpy(pszStr, &pszStr[i]);   /*- Shift string to the left */

      return(i);                          /* Return no of replacements */
}

/*
 /-------------------------------------\
|  STRTRIM                              |-----------------------------------|
|\-------------------------------------/
| Removes all leading and trailing blanks in a string.
| Blanks are defined with ISSPACE  (blank, tab, newline, return, formfeed,
| vertical tab = 0x09 - 0x0D + 0x20)
|
|
|-------------------------------------------------------------------------|
| CALL:
|    strtrim(&str);
|
| HEADER:
|    ctype.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|
| PROTOTYPE:
|    int _CfnTYPE strtrim(char *pszStr);
|
| RETURN VALUE:
|    iBlank      : No of removed blanks
|
| MODULE:
|    strtrim.c
|-------------------------------------------------------------------------|
|
|
|-------------------------------------------------------------------------|
|1992-11-09/Erik Bachmann
\-------------------------------------------------------------------------|*/

int _CfnTYPE strtrim(char *pszStr)
{
      int   iBlank;

      /*-------------------------------------------------*/

      iBlank  = strtrimr(pszStr);              /* Remove trailing blanks */
      iBlank += strtriml(pszStr);              /* Remove leading blanks */

      return(iBlank);
}


#ifdef      TEST

main()
{
      char  *strl = "     Leading blanks",
            *strr = "Traling blanks     ",
            *str  = "   Lead-&trailing  ";
      /*--------------------------------*/

      printf("\nBefore convertion:\n\t\"%s\"\n\t\"%s\"\n\t\"%s\"\n",
            strl, strr, str);

      strtrimr(strr);
      strtriml(strl);
      strtrim(str);
```

```
 201 |       printf("\nAfter convertion:\n\t\"%s\"\n\t\"%s\"\n\t\"%s\"\n",
 202 |                   strl, strr, str);
 203 |
 204 |       return(0);
 205 | }
 206 |
 207 | #endif
```

TEXT STATISTICS

```
  5787 characters
   207 lines
```

LEXICAL STATISTICS

```
    24 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     1 constants [character]
     6 constants [string]
     6 constants [numeric]
```

SYMBOL TABLE

```
    TEST           185
    _CfnTYPE        58     114     172
    ctype           16
    h       15      16
    i       60      64      66+     71      73     116     122+    126     128     129     131
    iBlank         174     178     179     181
    isspace         66     122
    j       60      64      73     116     120     122
    main           187
    printf         194     201
    pszStr          58      64      66      71     114     120     122     129+    172     178     179
    str            191     195     199     202
    strcpy         129
    string          15
    strl           189     195     198     202
    strlen          64     120
    strr           190     195     197     202
    strtrim        172     199
    strtriml       114     179     198
    strtrimr        58     178     197
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
* @(#)strtrimc.c
* @(#)strtrimcr
* @(#)      Removes all trailing occurences of specific characters in a string.
* @(#)strtrimcl
* @(#)      Removes all leading occurences of specific characters in a string.
* @(#)strtrimc
* @(#)      Removes all leading and traling occurences of specific characters
* @(#)      in a string.
*
*****************************************************************************
*@(#)1993 Erik Bachmann
*
* Released to public domain 27-Oct-95
*****************************************************************************/

#include <string.h>
#include "bacstd.h"

/*
 /-------------------------------------\
|  STRTRIMCR                           |-------------------------------------|
|\-------------------------------------/
|
| Removes all trailing occurences of specific characters in a string.
|
|
|-----------------------------------------------------------------------------|
| CALL:
|    strtrimcr(&str, ":;\\");
|
| HEADER:
|    ctype.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|    pszSet      : String with the characters to remove
|
|
| PROTOTYPE:
|    int _CfnTYPE strtrimcr(char *szStr, char *szSet);
|
| RETURN VALUE:
|    j-i         : No of removed characters
|
| MODULE:
|    strtrim.c
|-----------------------------------------------------------------------------|
|
|
|-----------------------------------------------------------------------------|
|1992-11-09/Erik Bachmann
\-----------------------------------------------------------------------------|*/

int _CfnTYPE strtrimcr(char *szStr, char *szSet)
{
      int   i, j;                                   /* Locale counters */

      /*-------------------------------------------------*/

      j = i = strlen(szStr) - 1;               /* Find length of string */

      while (strrchr(szSet, szStr[ i ])
                  && (0 <= i))
      {
            /* While string is terminated by one of the specified characters */
            szStr[ i-- ] = '\0';          /*- Replace character with '\0' */
      }

      return(j - i);    /* Return the difference between old and new length */
}

/*
 /-------------------------------------\
|  STRTRIMCL                           |-------------------------------------|
|\-------------------------------------/
|
| Removes all leading occurences of a specific character in a string.
|
|-----------------------------------------------------------------------------|
| CALL:
|    strtrimcl(&str, ";:\\");
|
| HEADER:
|    ctype.h
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszStr      : String to be converted
|    pszSet      : String with the characters to remove
|
| PROTOTYPE:
|    int _CfnTYPE strtrimcl(char *szStr, char *szSet);
```

```
101 | |
102 | | RETURN VALUE:
103 | |    i           : No of removed characters
104 | |
105 | | MODULE:
106 | |    strtrim.c
107 | |-------------------------------------------------------------------------|
108 | |
109 | |
110 | |-------------------------------------------------------------------------|
111 | |1992-11-09/Erik Bachmann
112 |  \-------------------------------------------------------------------------|*/
113 |
114 | int _CfnTYPE strtrimcl(char *szStr, char *szSet)
115 | {
116 |       int   i = 0, j;
117 |
118 |       /*-----------------------------------------------*/
119 |
120 |       j = strlen(szStr) - 1;                    /* Find length of string */
121 |
122 |       while (strrchr(szSet, szStr[ i ])
123 |                    && (i <= j))
124 |       {
125 |             /* While first character in string matches tag */
126 |
127 |             i++;                    /*- Count no of removed chars */
128 |       }
129 |
130 |       if (0 < i)                    /* IF there were matches */
131 |             strcpy(szStr, &szStr[ i ]);         /*- shift string to the left */
132 |
133 |       return(i);                    /* Return no of matching chars */
134 | }
135 |
136 | /*
137 |  /-------------------------------------\
138 | |  STRTRIMC                            |-----------------------------------|
139 | |\-------------------------------------/
140 | |
141 | | Removes all leading and trailing occurences of a specific character in
142 | |  a string.
143 | |
144 | |-------------------------------------------------------------------------|
145 | | CALL:
146 | |    strtrimc(&str, ";:\\");
147 | |
148 | | HEADER:
149 | |    ctype.h
150 | |
151 | | GLOBALE VARIABLES:
152 | |    %
153 | |
154 | | ARGUMENTS:
155 | |    pszStr      : String to be converted
156 | |    pszSet      : String with the characters to remove
157 | |
158 | | PROTOTYPE:
159 | |    int _CfnTYPE strtrimc(char *szStr, char *szSet);
160 | |
161 | | RETURN VALUE:
162 | |    iStatusFlag : No of removed characters
163 | |
164 | | MODULE:
165 | |    strtrimc.c
166 | |-------------------------------------------------------------------------|
167 | |
168 | |
169 | |
170 | |-------------------------------------------------------------------------|
171 | |1992-11-09/Erik Bachmann
172 |  \-------------------------------------------------------------------------|*/
173 |
174 | int _CfnTYPE strtrimc(char *szStr, char *szSet)
175 | {
176 |       int   iStatusFlag;
177 |
178 |       /*-----------------------------------------------*/
179 |
180 |       iStatusFlag =  strtrimcl(szStr, szSet);
181 |       iStatusFlag += strtrimcr(szStr, szSet);
182 |
183 |       return(iStatusFlag);
184 | }
185 |
186 |  /*
187 |  /-------------------------------------\
188 | |  REP_LAST_CHAR                       |-----------------------------------|
189 | |\-------------------------------------/
190 | |
191 | | Replaces the last char on match with another specified char.
192 | |
193 | |
194 | |-------------------------------------------------------------------------|
195 | | CALL:
196 | |    rep_last_char(str, '\n', '\0');
197 | |
198 | | HEADER:
199 | |    string.h
200 | |
```

```
201 | | GLOBALE VARIABLES:
202 | |    %
203 | |
204 | | ARGUMENTS:
205 | |    pszStr      : String to be converted
206 | |    cChar1      : Character to replace
207 | |    cChar2      : Character to replace with
208 | |
209 | |
210 | | PROTOTYPE:
211 | |    int _CfnTYPE rep_last_char(char *szStr, char cChar1, char cChar2);
212 | |
213 | | RETURN VALUE:
214 | |    int         : Stringlength
215 | |
216 | | MODULE:
217 | |    strtrim.c
218 | |-----------------------------------------------------------------------------|
219 | |
220 | |
221 | |-----------------------------------------------------------------------------|
222 | |1992-11-09/Erik Bachmann
223 | \-----------------------------------------------------------------------------|*/
224 |
225 | int _CfnTYPE rep_last_char(char *pszStr, char cChar1, char cChar2)
226 | {
227 |       int   i;
228 |
229 |       /*--------------------------------*/
230 |
231 |       i = strlen(pszStr) - 1;
232 |
233 |       if (pszStr[ i ] == cChar1)
234 |             pszStr[ i ] = cChar2;
235 |
236 |       return(i);
237 | }     /*** rep_last_char() ***/
238 |
239 | #ifdef TEST
240 |
241 | main()
242 | {
243 |       char  *strl = "xxyzxxxLeading x",
244 |                     *strr = "x Traling xxyzxxx",
245 |                     *str  = "xxzyxLead-&trailingxzxyx";
246 |       /*--------------------------------*/
247 |
248 |       printf("\nBefore convertion:\n\t\"%s\"\n\t\"%s\"\n\t\"%s\"\n",
249 |                     strl, strr, str);
250 |
251 |       strtrimcr(strr, "xyz");
252 |       strtrimcl(strl, "xyz");
253 |       strtrimc( str, "xyz");
254 |
255 |       printf("\nAfter convertion:\n\t\"%s\"\n\t\"%s\"\n\t\"%s\"\n",
256 |                     strl, strr, str);
257 |
258 |       return(0);
259 | }
260 |
261 | #endif /* TEST */
```

TEXT STATISTICS

```
  7158 characters
   261 lines
```

LEXICAL STATISTICS

```
    24 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     1 constants [character]
     9 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    TEST           239
    _CfnTYPE        60     114     174     225
    cChar1         225     233
    cChar2         225     234
    h       19
    i       62      66      68      69      72      75     116     122     123     127     130     131
           133     227     231     233     234     236
    iStatusFlag            176     180     181     183
    j       62      66      75     116     120     123
    main           241
    printf         248     255
    pszStr         225     231     233     234
    rep_last_char          225
    str            245     249     253     256
    strcpy         131
    string          19
    strl           243     249     252     256
    strlen          66     120     231
    strr           244     249     251     256
    strrchr         68     122
    strtrimc       174     253
    strtrimcl      114     180     252
    strtrimcr       60     181     251
    szSet           60      68     114     122     174     180     181
    szStr           60      66      68      72     114     120     122     131+    174     180     181
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************
 * Generic structure <> disk file manipulations. These functions
 * form a basic template for reading and writing structures to a
 * sequential file. This template is probably most useful for files
 * with 500 or less records and eliminates the need for a more
 * elaborate file handler such as C-Tree, DB-Vista, Mix etc.
 * Routines to put data in the struct is out of scope here.
 * Written by Lynn Nash 8/28/91 and donated to the public domain.
 */
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include "sniptype.h"
#include "strucfil.h"

/*-------------------- general globals ---------------------*/

static long cur_rec = 0;      /* the current record number */
static FILE *fsptr = NULL;    /* fixed record data file pointer */

/* if file exists open in read/write mode else create file */

FILE * open_file(char *filename)
{
      if (access(filename, 0) == 0)
            fsptr = fopen(filename, "rb+");
      else  fsptr = fopen(filename, "wb+");
      return fsptr;                       /* return the file pointer */
}

/* add new records to the data file */

int datadd(void)
{
      if (fsptr)
      {
            if (fseek(fsptr, 0L, SEEK_END) != 0)
                  return Error_;  /* seek failure */
            rec.delete_flag = 0; /* active record tag */
            rec.recordnum = (int) (ftell(fsptr) /
                  (long) sizeof(struct blackbook));
            if (fwrite(&rec, sizeof(struct blackbook), 1, fsptr) != 1)
            {
                  return Error_; /* write error */
            }
            else
            {
                  /* put your clean up code here */
                  return Success_;
            }
      }
      return Error_;
}

/* tag the last record read in the file as deleted */

int data_delete(void)
{
      if (fsptr)
      {
            if (fseek(fsptr, (long) sizeof(struct blackbook) * -1L,
                  SEEK_CUR) != 0)
            {
                  return Error_;
            }
            rec.delete_flag = -1;   /* tag the record as deleted */
            if (fwrite(&rec, sizeof(struct blackbook), 1, fsptr) != 1)
                  return Error_;
            else  return Success_;
      }
      return Error_;
}

/* read a random structure. If successful the global cur_rec will
 * contain the number of the last record read & it can be compared
 * to the number in the struct as a double check (belt & suspenders)
 */

int data_read(long recnum)
{
      if (fseek(fsptr, (long) sizeof(struct blackbook) * recnum,
            SEEK_SET) != 0)
      {
            return Error_;
      }
      cur_rec = recnum; /* keep tabs on record pointer in global */

      /* now read the record into save struct*/

      if (fread(&oldrec, sizeof(struct blackbook), 1, fsptr) != 1)
      {
            return Error_;
      }
      else                          /* copy save struct to edit struct */
      {
            memcpy(&rec, &oldrec, sizeof(struct blackbook));
            return Success_;
      }
```

```
}

/* rewrite the last read record back to disk */

int data_update(void)
{
      if (memcmp(&rec, &oldrec, sizeof(struct blackbook)) == 0)
            return True_;  /* no update needed */

      /* back up one record before writing */

      if (fseek(fsptr, (long) sizeof(struct blackbook) * -1L,
            SEEK_CUR) != 0)
      {
            return Error_; /* seek error */
      }

      /* now write the record */

      if (fwrite(&rec, sizeof(struct blackbook), 1, fsptr) != 1)
            return Error_; /* write error */
      return Success_;
}

/* get the next valid record in the file */

int read_forward(void)
{
      do
      {
            cur_rec++; /* upcount the record number */
            if (data_read(cur_rec) != 0)
            {
                  cur_rec--; /* decrement the record number */
                  return Error_;
            }
      } while (oldrec.delete_flag != 0); /* record read was deleted */
      return Success_;
}

/* get the previous valid record in the file */

int read_backward(void)
{
      do
      {
            cur_rec--;  /* decrement the record number */
            if (cur_rec >= 0)
            {
                  if ( data_read(cur_rec) != 0 )
                  {
                        cur_rec++; /* increment the record number */
                        return Error_;
                  }
            }
      } while (oldrec.delete_flag != 0); /* record read was deleted */
      return Success_;
}
```

TEXT STATISTICS

```
  4740 characters
   158 lines
```

LEXICAL STATISTICS

```
    32 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     4 constants [string]
    26 constants [numeric]
```

SYMBOL TABLE

```
    Error_          40      46      54      66      70      73      86      94     115     121     135
         153
    FILE            21      25
    NULL            21
    SEEK_CUR        64     113
    SEEK_END        39
    SEEK_SET        84
    Success_        51      71      99     122     138     157
    True_          108
    access          27
    blackbook       43      44      63      69      83      92      98     107     112     120
    cur_rec         20      88     131     132     134     147     148     150     152
    data_delete             59
    data_read       81     132     150
    data_update            105
    dataadd         35
    delete_flag             41      68     137     156
    filename        25      27      28      29
    fopen           28      29
    fread           92
    fseek           39      63      83     112
    fsptr           21      28      29      30      37      39      42      44      61      63      69
           83       92     112     120
    ftell           42
    fwrite          44      69     120
    h       12      13      14
    io              12
    memcmp         107
    memcpy          98
    oldrec          92      98     107     137     156
    open_file       25
    read_backward          143
    read_forward           127
    rec             41      42      44      68      69      98     107     120
    recnum          81      83      88
    recordnum       42
    stdlib          14
    string          13
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  SNIPPETS header file for STRUCFIL.C
 5 | */
 6 |
 7 | #ifndef STRUCFIL__H
 8 | #define STRUCFIL__H
 9 |
10 | #include <stdio.h>
11 |
12 | /* make sure the record structure is byte aligned */
13 |
14 | #if defined(_MSC_VER) || defined(_QC) || defined(__WATCOMC__)
15 |  #pragma pack(1)
16 | #elif defined(__ZTC__)
17 |  #pragma ZTC align 1
18 | #elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
19 |  #pragma option -a-
20 | #endif
21 |
22 | static struct blackbook {
23 |     int delete_flag;        /* 0 = active  -1 = deleted */
24 |     int recordnum;          /* a sequential number in the file */
25 |    /* The data fields in asciiz.  */
26 |     char firstname[11];
27 |     char lastname[16];
28 |     char addr[26];
29 |     char city[16];
30 |     char state[3];
31 |     char zip[10];
32 |     char phone[11];
33 | } rec, oldrec;             /* 97 byte record  * 2 */
34 |
35 | /* set structure alignment to default */
36 |
37 | #if defined (_MSC_VER) || defined(_QC) || defined(__WATCOMC__)
38 |  #pragma pack()
39 | #elif defined (__ZTC__)
40 |  #pragma ZTC align
41 | #elif defined(__TURBOC__) && (__TURBOC__ > 0x202)
42 |  #pragma option -a.
43 | #endif
44 |
45 | FILE * open_file(char *filename);
46 | int    datadd(void);
47 | int    data_delete(void);
48 | int    data_read(long recnum);
49 | int    data_update(void);
50 | int    read_forward(void);
51 | int    read_backward(void);
52 |
53 | #endif /* STRUCFIL__H */
```

TEXT STATISTICS

```
  1303 characters
    53 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
    18 preprocessor instructions
     0 constants [character]
     0 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    FILE            45
    STRUCFIL__H              7       8
    ZTC             17      40
    _MSC_VER        14      37
    _QC             14      37
    __TURBOC__              18+     41+
    __WATCOMC__             14      37
    __ZTC__         16      39
    a       19      42
    addr            28
    align           17      40
    blackbook       22
    city            29
    data_delete             47
    data_read       48
    data_update             49
    datadd          46
    defined         14+     16      18      37+     39      41
    delete_flag             23
    filename        45
    firstname       26
    h       10
    lastname        27
    oldrec          33
    open_file       45
    option          19      42
    pack            15      38
    phone           32
    read_backward           51
    read_forward            50
    rec             33
    recnum          48
    recordnum       24
    state           30
    stdio           10
    zip             31
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Portable, public domain replacements for strupr() & strlwr() by Bob Stout
*/

#include <ctype.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *strupr(char *string)
{
      char *s;

      if (string)
      {
            for (s = string; *s; ++s)
                  *s = toupper(*s);
      }
      return string;
}

char *strlwr(char *string)
{
      char *s;

      if (string)
      {
            for (s = string; *s; ++s)
                  *s = tolower(*s);
      }
      return string;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
      if (argc < 2)
      {
            puts("Usage: STRUPR string1 [...string2 [...stringN]]");
            return EXIT_FAILURE;
      }
      while (--argc)
      {
            printf("Original = \"%s\"\n", *++argv);
            printf("strupr() = \"%s\"\n", strupr(*argv));
            printf("strlwr() = \"%s\"\n", strlwr(*argv));
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1175 characters
    63 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     6 constants [string]
     1 constants [numeric]

SYMBOL TABLE

    EXIT_FAILURE            52
    EXIT_SUCCESS            60
    TEST            42
    __cplusplus             10+     38+
    argc            47      49      54
    argv            47      56      57      58
    ctype            7
    defined         10      38
    h        7      44      45
    main            47
    printf          56      57      58
    puts            51
    s       16      20+     21+     28      32+     33+
    stdio           44
    stdlib          45
    string          14      18      20      23      26      30      32      35
    strlwr          26      58
    strupr          14      57
    tolower         33
    toupper         21
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  STUB.C - Utility to truncate files
**
**      STUB is used with MAKE utilities which lack the ability to timestamp
**      library object modules to reduce disk space requirements. After
**      compiling and building your library or executable, run "STUB *.OBJ"
**      to truncate all object files to zero length. The files' time and
**      date stamps will remain unchanged. By doing this, your make utility
**      will still know which modules are out of date even though the files
**      themselves have all been truncated to zero length. STUB also supports
**      the standard response file format so you can use your linker response
**      files to direct the files to be truncated.
**
**  public domain by Bob Stout
**
**  Notes: To expand command line arguments with wildcards,
**         TC/TC++/BC++  - Link in WILDARGS.OBJ.
**         MSC/QC        - Link in SETARGV.OBJ.
**         ZTC/C++       - Link in _MAINx.OBJ, where 'x' is the memory model.
**         Watcom C/C++  - Compile & link with WILDARGV.C
**
**         Allows file list(s) using standard "@file_list_name" convention.
*/

#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <io.h>
#include <fcntl.h>
#include "sniptype.h"

int fd;

void truncate(char *);

int main(int argc, char **argv)
{
      int i;

      if (2 > argc)
      {
            puts("Usage: STUB filespec [...filespec]");
            puts("where: filespec = fully-specified file name, or");
            puts("       filespec = wildcard-specified file name, or");
            puts("       filespec = response file name, e.g. \"@FILE.LST\"");
            return 1;
      }

      for (i = 1; i < argc; ++i)    /* Scan for simple file specs       */
      {
            if ('@' == *argv[i])
                  continue;
            else  truncate(argv[i]);
      }
      for (i = 1; i < argc; ++i)    /* Scan for response file specs     */
      {
            if ('@' == *argv[i])
            {
                  FILE *fp;
                  char buf[256], *ptr = &argv[i][1];

                  if (NULL == (fp = fopen(ptr, "r")))
                  {
                        printf("\aSTUB: Error opening %s\n", ptr);
                        return -1;
                  }
                  while (NULL != fgets(buf, 255, fp))
                  {
                        LAST_CHAR(buf) = '\0';  /* Strip '\n'           */
                        truncate(buf);
                  }
                  fclose(fp);
            }
      }
      return 0;
}

/*
**  The actual truncation function
*/

#ifdef __ZTC__
 #define GETFTIME dos_getftime
 #define SETFTIME dos_setftime
#else
 #define GETFTIME _dos_getftime
 #define SETFTIME _dos_setftime
#endif

#ifdef __WATCOMC__
 typedef unsigned short FTIME_T;
#else
 typedef unsigned int FTIME_T;
#endif

void truncate(char *fname)
{
#ifdef __TURBOC__
```

```
 101 |             struct ftime Ftime;
 102 | #else
 103 |             unsigned short date, time;
 104 | #endif
 105 |
 106 |             fd = open(fname, O_WRONLY);
 107 |
 108 | #ifdef __TURBOC__                       /* Save the time/date         */
 109 |             getftime(fd, &Ftime);
 110 | #else
 111 |             GETFTIME(fd, (FTIME_T *)&date, (FTIME_T *)&time);
 112 | #endif
 113 |
 114 |             chsize(fd, 0L);             /* Truncate the file          */
 115 |
 116 | #ifdef __TURBOC__                       /* Restore the time/date      */
 117 |             setftime(fd, &Ftime);
 118 | #else
 119 |             SETFTIME(fd, (FTIME_T)date, (FTIME_T)time);
 120 | #endif
 121 |             close(fd);
 122 | }
```

TEXT STATISTICS

```
  3615 characters
   122 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
    25 preprocessor instructions
     3 constants [character]
     7 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    FILE            61
    FTIME_T         93      95     111+    119+
    Ftime          101     109     117
    GETFTIME        85      88     111
    LAST_CHAR       71
    NULL            64      69
    O_WRONLY       106
    SETFTIME        86      89     119
    __TURBOC__             100     108     116
    __WATCOMC__             92
    __ZTC__         84
    _dos_getftime           88
    _dos_setftime           89
    argc            38      42      51      57
    argv            38      53      55      59      62
    buf             62      69      71      72
    chsize         114
    close          121
    date           103     111     119
    dos             29
    dos_getftime            85
    dos_setftime            86
    fclose          74
    fcntl           31
    fd              34     106     109     111     114     117     119     121
    fgets           69
    fname           98     106
    fopen           64
    fp              61      64      69      74
    ftime          101
    getftime       109
    h       27      28      29      30      31
    i       40      51+     53      55      57+     59      62
    io              30
    main            38
    open           106
    printf          66
    ptr             62      64      66
    puts            44      45      46      47
    setftime       117
    stdio           27
    string          28
    time           103     111     119
    truncate        36      55      72      98
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*****************************************************************************/
  4 | /*  Global (public) headers look like this                                   */
  5 | /*                                                                           */
  6 | /*  foo()                                                                    */
  7 | /*  Parameters: description of each passed datum                             */
  8 | /*  Returns: description of return value & significance                      */
  9 | /*  Side effects: complete details                                           */
 10 | /*  Notes: extra info                                                        */
 11 | /*****************************************************************************/
 12 |
 13 | /*
 14 | ** Static (local) headers look like this
 15 | **
 16 | ** Brief description
 17 | */
 18 |
 19 | int foo(void)   /* use ANSI prototypes on every compiler that supports 'em */
 20 | {
 21 |       /* First of all, block comments (like this one) are preferable to    */
 22 |       /* trailing (in-line) comments as in the following examples.         */
 23 |
 24 |       int local1, local2;               /* local variable declarations are
 25 |                                            always followed by a blank line */
 26 |
 27 |       do_stuff();
 28 |       if (bar(local1))
 29 |       {     /* long comments here     *//* this lines up with -----------+ */
 30 |             char *local3;               /* autos declared close to use   | */
 31 |                                         /*                               | */
 32 |             do_more_stuff();           /* (everything else indented)    | */
 33 |             local2 = strlen(local3);    /*                               | */
 34 |       }                                 /* this <------------------------+ */
 35 |       else  local2 = fubar();           /* using tab >= 6, else's line up  */
 36 |       return local2;
 37 | }                                       /* no question where functions end!*/
```

TEXT STATISTICS

```
  1931 characters
    37 lines
```

LEXICAL STATISTICS

```
    24 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    bar             28
    do_more_stuff           32
    do_stuff        27
    foo             19
    fubar           35
    local1          24      28
    local2          24      33      35      36
    local3          30      33
    strlen          33
```

```
/* +++Date last modified: 05-Jul-1997 */

/*

SUNRISET.C - computes Sun rise/set times, start/end of twilight, and
             the length of the day at any date and latitude

Written as DAYLEN.C, 1989-08-16

Modified to SUNRISET.C, 1992-12-01

(c) Paul Schlyter, 1989, 1992

Released to the public domain by Paul Schlyter, December 1992

*/


#include <stdio.h>
#include <math.h>


/* A macro to compute the number of days elapsed since 2000 Jan 0.0 */
/* (which is equal to 1999 Dec 31, 0h UT)                           */

#define days_since_2000_Jan_0(y,m,d) \
    (367L*(y)-((7*((y)+(((m)+9)/12)))/4)+((275*(m))/9)+(d)-730530L)

/* Some conversion factors between radians and degrees */

#ifndef PI
 #define PI        3.141592653589793238
#endif

#define RADEG     ( 180.0 / PI )
#define DEGRAD    ( PI / 180.0 )

/* The trigonometric functions in degrees */

#define sind(x)  sin((x)*DEGRAD)
#define cosd(x)  cos((x)*DEGRAD)
#define tand(x)  tan((x)*DEGRAD)

#define atand(x)    (RADEG*atan(x))
#define asind(x)    (RADEG*asin(x))
#define acosd(x)    (RADEG*acos(x))
#define atan2d(y,x) (RADEG*atan2(y,x))


/* Following are some macros around the "workhorse" function __daylen__ */
/* They mainly fill in the desired values for the reference altitude    */
/* below the horizon, and also selects whether this altitude should     */
/* refer to the Sun's center or its upper limb.                         */


/* This macro computes the length of the day, from sunrise to sunset. */
/* Sunrise/set is considered to occur when the Sun's upper limb is    */
/* 35 arc minutes below the horizon (this accounts for the refraction */
/* of the Earth's atmosphere).                                        */
#define day_length(year,month,day,lon,lat)  \
        __daylen__( year, month, day, lon, lat, -35.0/60.0, 1 )

/* This macro computes the length of the day, including civil twilight. */
/* Civil twilight starts/ends when the Sun's center is 6 degrees below  */
/* the horizon.                                                         */
#define day_civil_twilight_length(year,month,day,lon,lat)  \
        __daylen__( year, month, day, lon, lat, -6.0, 0 )

/* This macro computes the length of the day, incl. nautical twilight.  */
/* Nautical twilight starts/ends when the Sun's center is 12 degrees    */
/* below the horizon.                                                   */
#define day_nautical_twilight_length(year,month,day,lon,lat)  \
        __daylen__( year, month, day, lon, lat, -12.0, 0 )

/* This macro computes the length of the day, incl. astronomical twilight. */
/* Astronomical twilight starts/ends when the Sun's center is 18 degrees   */
/* below the horizon.                                                      */
#define day_astronomical_twilight_length(year,month,day,lon,lat)  \
        __daylen__( year, month, day, lon, lat, -18.0, 0 )


/* This macro computes times for sunrise/sunset.                      */
/* Sunrise/set is considered to occur when the Sun's upper limb is    */
/* 35 arc minutes below the horizon (this accounts for the refraction */
/* of the Earth's atmosphere).                                        */
#define sun_rise_set(year,month,day,lon,lat,rise,set)  \
        __sunriset__( year, month, day, lon, lat, -35.0/60.0, 1, rise, set )

/* This macro computes the start and end times of civil twilight.       */
/* Civil twilight starts/ends when the Sun's center is 6 degrees below  */
/* the horizon.                                                         */
#define civil_twilight(year,month,day,lon,lat,start,end)  \
        __sunriset__( year, month, day, lon, lat, -6.0, 0, start, end )

/* This macro computes the start and end times of nautical twilight.    */
/* Nautical twilight starts/ends when the Sun's center is 12 degrees    */
/* below the horizon.                                                   */
#define nautical_twilight(year,month,day,lon,lat,start,end)  \
        __sunriset__( year, month, day, lon, lat, -12.0, 0, start, end )

```

```
/* This macro computes the start and end times of astronomical twilight.   */
/* Astronomical twilight starts/ends when the Sun's center is 18 degrees   */
/* below the horizon.                                                      */
#define astronomical_twilight(year,month,day,lon,lat,start,end)  \
        __sunriset__( year, month, day, lon, lat, -18.0, 0, start, end )


/* Function prototypes */

double __daylen__( int year, int month, int day, double lon, double lat,
                   double altit, int upper_limb );

int __sunriset__( int year, int month, int day, double lon, double lat,
                  double altit, int upper_limb, double *rise, double *set );

void sunpos( double d, double *lon, double *r );

void sun_RA_dec( double d, double *RA, double *dec, double *r );

double revolution( double x );

double rev180( double x );

double GMST0( double d );



/* A small test program */

main()
{
      int year,month,day;
      double lon, lat;
      double daylen, civlen, nautlen, astrlen;
      double rise, set, civ_start, civ_end, naut_start, naut_end,
             astr_start, astr_end;
      int    rs, civ, naut, astr;
      char buf[80];

      printf( "Longitude (+ is east) and latitude (+ is north) : " );
      fgets(buf, 80, stdin);
      sscanf(buf, "%lf %lf", &lon, &lat );

      for(;;)
      {
            printf( "Input date ( yyyy mm dd ) (ctrl-C exits): " );
            fgets(buf, 80, stdin);
            sscanf(buf, "%d %d %d", &year, &month, &day );

            daylen  = day_length(year,month,day,lon,lat);
            civlen  = day_civil_twilight_length(year,month,day,lon,lat);
            nautlen = day_nautical_twilight_length(year,month,day,lon,lat);
            astrlen = day_astronomical_twilight_length(year,month,day,
                  lon,lat);

            printf( "Day length:                %5.2f hours\n", daylen );
            printf( "With civil twilight        %5.2f hours\n", civlen );
            printf( "With nautical twilight     %5.2f hours\n", nautlen );
            printf( "With astronomical twilight  %5.2f hours\n", astrlen );
            printf( "Length of twilight: civil   %5.2f hours\n",
                  (civlen-daylen)/2.0);
            printf( "                  nautical  %5.2f hours\n",
                  (nautlen-daylen)/2.0);
            printf( "              astronomical  %5.2f hours\n",
                  (astrlen-daylen)/2.0);

            rs   = sun_rise_set         ( year, month, day, lon, lat,
                                          &rise, &set );
            civ  = civil_twilight       ( year, month, day, lon, lat,
                                          &civ_start, &civ_end );
            naut = nautical_twilight    ( year, month, day, lon, lat,
                                          &naut_start, &naut_end );
            astr = astronomical_twilight( year, month, day, lon, lat,
                                          &astr_start, &astr_end );

            printf( "Sun at south %5.2fh UT\n", (rise+set)/2.0 );

            switch( rs )
            {
                case 0:
                    printf( "Sun rises %5.2fh UT, sets %5.2fh UT\n",
                             rise, set );
                    break;
                case +1:
                    printf( "Sun above horizon\n" );
                    break;
                case -1:
                    printf( "Sun below horizon\n" );
                    break;
            }

            switch( civ )
            {
                case 0:
                    printf( "Civil twilight starts %5.2fh, "
                            "ends %5.2fh UT\n", civ_start, civ_end );
                    break;
                case +1:
                    printf( "Never darker than civil twilight\n" );
                    break;
```

```
                case -1:
                    printf( "Never as bright as civil twilight\n" );
                    break;
            }

            switch( naut )
            {
                case 0:
                    printf( "Nautical twilight starts %5.2fh, "
                            "ends %5.2fh UT\n", naut_start, naut_end );
                    break;
                case +1:
                    printf( "Never darker than nautical twilight\n" );
                    break;
                case -1:
                    printf( "Never as bright as nautical twilight\n" );
                    break;
            }

            switch( astr )
            {
                case 0:
                    printf( "Astronomical twilight starts %5.2fh, "
                            "ends %5.2fh UT\n", astr_start, astr_end );
                    break;
                case +1:
                    printf( "Never darker than astronomical twilight\n" );
                    break;
                case -1:
                    printf( "Never as bright as astronomical twilight\n" );
                    break;
            }
      return 0;
      }
}

/* The "workhorse" function for sun rise/set times */

int __sunriset__( int year, int month, int day, double lon, double lat,
                  double altit, int upper_limb, double *trise, double *tset )
/*********************************************************************/
/* Note: year,month,date = calendar date, 1801-2099 only.             */
/*       Eastern longitude positive, Western longitude negative       */
/*       Northern latitude positive, Southern latitude negative       */
/*       The longitude value IS critical in this function!            */
/*       altit = the altitude which the Sun should cross              */
/*               Set to -35/60 degrees for rise/set, -6 degrees       */
/*               for civil, -12 degrees for nautical and -18          */
/*               degrees for astronomical twilight.                   */
/*         upper_limb: non-zero -> upper limb, zero -> center         */
/*               Set to non-zero (e.g. 1) when computing rise/set     */
/*               times, and to zero when computing start/end of       */
/*               twilight.                                            */
/*        *rise = where to store the rise time                        */
/*        *set  = where to store the set  time                        */
/*                Both times are relative to the specified altitude,  */
/*                and thus this function can be used to compute       */
/*                various twilight times, as well as rise/set times   */
/* Return value:  0 = sun rises/sets this day, times stored at        */
/*                    *trise and *tset.                               */
/*               +1 = sun above the specified "horizon" 24 hours.     */
/*                    *trise set to time when the sun is at south,    */
/*                    minus 12 hours while *tset is set to the south  */
/*                    time plus 12 hours. "Day" length = 24 hours     */
/*               -1 = sun is below the specified "horizon" 24 hours   */
/*                    "Day" length = 0 hours, *trise and *tset are    */
/*                    both set to the time when the sun is at south.  */
/*                                                                    */
/**********************************************************************/
{
      double  d,  /* Days since 2000 Jan 0.0 (negative before) */
      sr,         /* Solar distance, astronomical units */
      sRA,        /* Sun's Right Ascension */
      sdec,       /* Sun's declination */
      sradius,    /* Sun's apparent radius */
      t,          /* Diurnal arc */
      tsouth,     /* Time when Sun is at south */
      sidtime;    /* Local sidereal time */

      int rc = 0; /* Return cde from function - usually 0 */

      /* Compute d of 12h local mean solar time */
      d = days_since_2000_Jan_0(year,month,day) + 0.5 - lon/360.0;

      /* Compute local sidereal time of this moment */
      sidtime = revolution( GMST0(d) + 180.0 + lon );

      /* Compute Sun's RA + Decl at this moment */
      sun_RA_dec( d, &sRA, &sdec, &sr );

      /* Compute time when Sun is at south - in hours UT */
      tsouth = 12.0 - rev180(sidtime - sRA)/15.0;

      /* Compute the Sun's apparent radius, degrees */
      sradius = 0.2666 / sr;

      /* Do correction to upper limb, if necessary */
      if ( upper_limb )
            altit -= sradius;
```

```
301 |
302 |       /* Compute the diurnal arc that the Sun traverses to reach */
303 |       /* the specified altitude altit: */
304 |       {
305 |             double cost;
306 |             cost = ( sind(altit) - sind(lat) * sind(sdec) ) /
307 |                   ( cosd(lat) * cosd(sdec) );
308 |             if ( cost >= 1.0 )
309 |                   rc = -1, t = 0.0;       /* Sun always below altit */
310 |             else if ( cost <= -1.0 )
311 |                   rc = +1, t = 12.0;      /* Sun always above altit */
312 |             else
313 |                   t = acosd(cost)/15.0;   /* The diurnal arc, hours */
314 |       }
315 |
316 |       /* Store rise and set times - in hours UT */
317 |       *trise = tsouth - t;
318 |       *tset  = tsouth + t;
319 |
320 |       return rc;
321 | }  /* __sunriset__ */
322 |
323 |
324 |
325 | /* The "workhorse" function */
326 |
327 |
328 | double __daylen__( int year, int month, int day, double lon, double lat,
329 |                    double altit, int upper_limb )
330 | /**********************************************************************/
331 | /* Note: year,month,date = calendar date, 1801-2099 only.             */
332 | /*       Eastern longitude positive, Western longitude negative       */
333 | /*       Northern latitude positive, Southern latitude negative       */
334 | /*       The longitude value is not critical. Set it to the correct   */
335 | /*       longitude if you're picky, otherwise set to to, say, 0.0     */
336 | /*       The latitude however IS critical - be sure to get it correct */
337 | /*       altit = the altitude which the Sun should cross              */
338 | /*               Set to -35/60 degrees for rise/set, -6 degrees       */
339 | /*               for civil, -12 degrees for nautical and -18          */
340 | /*               degrees for astronomical twilight.                   */
341 | /*         upper_limb: non-zero -> upper limb, zero -> center         */
342 | /*               Set to non-zero (e.g. 1) when computing day length   */
343 | /*               and to zero when computing day+twilight length.      */
344 | /**********************************************************************/
345 | {
346 |       double  d,  /* Days since 2000 Jan 0.0 (negative before) */
347 |       obl_ecl,    /* Obliquity (inclination) of Earth's axis */
348 |       sr,         /* Solar distance, astronomical units */
349 |       slon,       /* True solar longitude */
350 |       sin_sdecl,  /* Sine of Sun's declination */
351 |       cos_sdecl,  /* Cosine of Sun's declination */
352 |       sradius,    /* Sun's apparent radius */
353 |       t;          /* Diurnal arc */
354 |
355 |       /* Compute d of 12h local mean solar time */
356 |       d = days_since_2000_Jan_0(year,month,day) + 0.5 - lon/360.0;
357 |
358 |       /* Compute obliquity of ecliptic (inclination of Earth's axis) */
359 |       obl_ecl = 23.4393 - 3.563E-7 * d;
360 |
361 |       /* Compute Sun's position */
362 |       sunpos( d, &slon, &sr );
363 |
364 |       /* Compute sine and cosine of Sun's declination */
365 |       sin_sdecl = sind(obl_ecl) * sind(slon);
366 |       cos_sdecl = sqrt( 1.0 - sin_sdecl * sin_sdecl );
367 |
368 |       /* Compute the Sun's apparent radius, degrees */
369 |       sradius = 0.2666 / sr;
370 |
371 |       /* Do correction to upper limb, if necessary */
372 |       if ( upper_limb )
373 |             altit -= sradius;
374 |
375 |       /* Compute the diurnal arc that the Sun traverses to reach */
376 |       /* the specified altitude altit: */
377 |       {
378 |             double cost;
379 |             cost = ( sind(altit) - sind(lat) * sin_sdecl ) /
380 |                   ( cosd(lat) * cos_sdecl );
381 |             if ( cost >= 1.0 )
382 |                   t = 0.0;                      /* Sun always below altit */
383 |             else if ( cost <= -1.0 )
384 |                   t = 24.0;                     /* Sun always above altit */
385 |             else  t = (2.0/15.0) * acosd(cost); /* The diurnal arc, hours */
386 |       }
387 |       return t;
388 | }  /* __daylen__ */
389 |
390 |
391 | /* This function computes the Sun's position at any instant */
392 |
393 | void sunpos( double d, double *lon, double *r )
394 | /******************************************************/
395 | /* Computes the Sun's ecliptic longitude and distance */
396 | /* at an instant given in d, number of days since     */
397 | /* 2000 Jan 0.0.  The Sun's ecliptic latitude is not  */
398 | /* computed, since it's always very near 0.           */
399 | /******************************************************/
400 | {
```

```
401 |       double M,         /* Mean anomaly of the Sun */
402 |              w,         /* Mean longitude of perihelion */
403 |                         /* Note: Sun's mean longitude = M + w */
404 |              e,         /* Eccentricity of Earth's orbit */
405 |              E,         /* Eccentric anomaly */
406 |              x, y,      /* x, y coordinates in orbit */
407 |              v;         /* True anomaly */
408 |
409 |       /* Compute mean elements */
410 |       M = revolution( 356.0470 + 0.9856002585 * d );
411 |       w = 282.9404 + 4.70935E-5 * d;
412 |       e = 0.016709 - 1.151E-9 * d;
413 |
414 |       /* Compute true longitude and radius vector */
415 |       E = M + e * RADEG * sind(M) * ( 1.0 + e * cosd(M) );
416 |            x = cosd(E) - e;
417 |       y = sqrt( 1.0 - e*e ) * sind(E);
418 |       *r = sqrt( x*x + y*y );              /* Solar distance */
419 |       v = atan2d( y, x );                  /* True anomaly */
420 |       *lon = v + w;                        /* True solar longitude */
421 |       if ( *lon >= 360.0 )
422 |            *lon -= 360.0;                   /* Make it 0..360 degrees */
423 | }
424 |
425 | void sun_RA_dec( double d, double *RA, double *dec, double *r )
426 | {
427 |       double lon, obl_ecl, x, y, z;
428 |
429 |       /* Compute Sun's ecliptical coordinates */
430 |       sunpos( d, &lon, r );
431 |
432 |       /* Compute ecliptic rectangular coordinates (z=0) */
433 |       x = *r * cosd(lon);
434 |       y = *r * sind(lon);
435 |
436 |       /* Compute obliquity of ecliptic (inclination of Earth's axis) */
437 |       obl_ecl = 23.4393 - 3.563E-7 * d;
438 |
439 |       /* Convert to equatorial rectangular coordinates - x is unchanged */
440 |       z = y * sind(obl_ecl);
441 |       y = y * cosd(obl_ecl);
442 |
443 |       /* Convert to spherical coordinates */
444 |       *RA = atan2d( y, x );
445 |       *dec = atan2d( z, sqrt(x*x + y*y) );
446 |
447 | }  /* sun_RA_dec */
448 |
449 |
450 | /******************************************************************/
451 | /* This function reduces any angle to within the first revolution */
452 | /* by subtracting or adding even multiples of 360.0 until the     */
453 | /* result is >= 0.0 and < 360.0                                   */
454 | /******************************************************************/
455 |
456 | #define INV360    ( 1.0 / 360.0 )
457 |
458 | double revolution( double x )
459 | /*****************************************/
460 | /* Reduce angle to within 0..360 degrees */
461 | /*****************************************/
462 | {
463 |       return( x - 360.0 * floor( x * INV360 ) );
464 | }  /* revolution */
465 |
466 | double rev180( double x )
467 | /*******************************************/
468 | /* Reduce angle to within +180..+180 degrees */
469 | /*******************************************/
470 | {
471 |       return( x - 360.0 * floor( x * INV360 + 0.5 ) );
472 | }  /* revolution */
473 |
474 |
475 | /******************************************************************/
476 | /* This function computes GMST0, the Greenwich Mean Sidereal Time  */
477 | /* at 0h UT (i.e. the sidereal time at the Greenwhich meridian at  */
478 | /* 0h UT).  GMST is then the sidereal time at Greenwich at any     */
479 | /* time of the day.  I've generalized GMST0 as well, and define it */
480 | /* as:  GMST0 = GMST - UT  --  this allows GMST0 to be computed at */
481 | /* other times than 0h UT as well.  While this sounds somewhat     */
482 | /* contradictory, it is very practical:  instead of computing      */
483 | /* GMST like:                                                      */
484 | /*                                                                 */
485 | /*  GMST = (GMST0) + UT * (366.2422/365.2422)                      */
486 | /*                                                                 */
487 | /* where (GMST0) is the GMST last time UT was 0 hours, one simply  */
488 | /* computes:                                                       */
489 | /*                                                                 */
490 | /*  GMST = GMST0 + UT                                              */
491 | /*                                                                 */
492 | /* where GMST0 is the GMST "at 0h UT" but at the current moment!   */
493 | /* Defined in this way, GMST0 will increase with about 4 min a     */
494 | /* day.  It also happens that GMST0 (in degrees, 1 hr = 15 degr)   */
495 | /* is equal to the Sun's mean longitude plus/minus 180 degrees!    */
496 | /* (if we neglect aberration, which amounts to 20 seconds of arc   */
497 | /* or 1.33 seconds of time)                                        */
498 | /*                                                                 */
499 | /******************************************************************/
500 |
```

```
501 | double GMST0( double d )
502 | {
503 |       double sidtim0;
504 |       /* Sidtime at 0h UT = L (Sun's mean longitude) + 180.0 degr  */
505 |       /* L = M + w, as defined in sunpos().  Since I'm too lazy to */
506 |       /* add these numbers, I'll let the C compiler do it for me.  */
507 |       /* Any decent C compiler will add the constants at compile   */
508 |       /* time, imposing no runtime or code overhead.               */
509 |       sidtim0 = revolution( ( 180.0 + 356.0470 + 282.9404 ) +
510 |                             ( 0.9856002585 + 4.70935E-5 ) * d );
511 |       return sidtim0;
512 | }  /* GMST0 */
```

TEXT STATISTICS

21862 characters
512 lines

LEXICAL STATISTICS

196 comments [std-C]
0 comments [C++]
24 preprocessor instructions
0 constants [character]
27 constants [string]
97 constants [numeric]

SYMBOL TABLE

```
    DEGRAD          36      40      41      42
    E      405     415     416     417
    GMST0           124     287     501
    INV360          456     463     471
    M      401     410     415+
    PI               31      32      35      36
    RA              118     425     444
    RADEG            35      44      45      46      47     415
    __daylen__                61      67      73      79     110     328
    __sunriset__              87      93      99     105     113     240
    acos             46
    acosd            46     313     385
    altit           111     114     241     300     306     329     373     379
    asin             45
    asind            45
    astr            137     173     220
    astr_end        136     174     224
    astr_start              136     174     224
    astrlen         134     153     159     165
    astronomical_twilight           104     173
    atan             44
    atan2            47
    atan2d           47     419     444     445
    atand            44
    buf             138     141     142     147     148
    civ             137     169     192
    civ_end         135     170     196
    civ_start       135     170     196
    civil_twilight           92     169
    civlen          134     151     157     161
    cos              41
    cos_sdecl       351     366     380
    cosd             41     307+    380     415     416     433     441
    cost            305     306     308     310     313     378     379     381     383     385
    d       26      27     116     118     124     272     284     287     290     346     356     359
           362     393     410     411     412     425     430     437     501     510
    day              60      61      66      67      72      73      78      79      86      87      92
            93      98      99     104     105     110     113     132     148     150     151     152
           153     167     169     171     173     240     284     328     356
    day_astronomical_twilight_length         78     153
    day_civil_twilight_length        66     151
    day_length               60     150
    day_nautical_twilight_length             72     152
    daylen          134     150     156     161     163     165
    days_since_2000_Jan_0             26     284     356
    dec             118     425     445
    e      404     412     415+    416     417+
    end              92      93      98      99     104     105
    fgets           141     147
    floor           463     471
    h       19      20
    lat              60      61      66      67      72      73      78      79      86      87      92
            93      98      99     104     105     110     113     133     142     150     151     152
           154     167     169     171     173     240     306     307     328     379     380
    lon              60      61      66      67      72      73      78      79      86      87      92
            93      98      99     104     105     110     113     116     133     142     150     151
           152     154     167     169     171     173     240     284     287     328     356     393
           420     421     422     427     430     433     434
    m       26      27+
    main            130
    math             20
    month            60      61      66      67      72      73      78      79      86      87      92
            93      98      99     104     105     110     113     132     148     150     151     152
           153     167     169     171     173     240     284     328     356
    naut            137     171     206
    naut_end        135     172     210
    naut_start              135     172     210
    nautical_twilight        98     171
    nautlen         134     152     158     163
    obl_ecl         347     359     365     427     437     440     441
    printf          140     146     156     157     158     159     160     162     164     176     181
           185     188     195     199     202     209     213     216     223     227     230
    r      116     118     393     418     425     430     433     434
    rc              281     309     311     320
    rev180          122     293     466
    revolution              120     287     410     458     509
    rise             86      87     114     135     168     176     182
    rs              137     167     178
    sRA             274     290     293
    sdec            275     290     306     307
    set              86      87     114     135     168     176     182
    sidtim0         503     509     511
    sidtime         279     287     293
    sin              40
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TABTRICKs.C - Demonstrates how to use printf() for columnar formatting
*/

#include <stdio.h>
#include <string.h>

#define putnum(i) putchar(i+'0')

main()
{
      char *firstname[] = { "Aloysius",   "Bob",   "Dennis",  NULL },
           *lastname[]  = { "Fuddrucker", "Stout", "Ritchie", NULL };
      int   score[]     = { -10,          70,      200,       0    },
            i, tabwidth;

      printf("%15sStudent Name%30s\n\n", "", "Test Score");
      for (i = 0; NULL != lastname[i]; ++i)
      {
            tabwidth = 36                              /* spaces to tab  */
                       -2                              /* allow for ", " */
                       -strlen(lastname[i]);           /* lastname space */
            printf("%15s%s, %-*s%3d\n",
                  "", lastname[i], tabwidth, firstname[i], score[i]);
      }

      /* print a ruler to make things clearer   */

      puts("");
      for (i = 0; i < 71; ++i)
      {
            if (i == 10 * (i / 10))
                  putnum(i / 10);
            else  putchar(' ');
      }
      puts("");
      for (i = 0; i < 71; ++i)
            putnum(i % 10);
      return 0;
}

/*

RESULTS:

               Student Name                    Test Score

               Fuddrucker, Aloysius               -10
               Stout, Bob                          70
               Ritchie, Dennis                    200

0         1         2         3         4         5         6         7
01234567890123456789012345678901234567890123456789012345678901234567890

*/
```

```
TEXT STATISTICS

  1654 characters
    57 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     2 constants [character]
    13 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    NULL            14      15      20
    firstname       14      26
    h        7       8
    i       10+     17      20+     24      26+     32+     34+     35      39+     40
    lastname        15      20      24      26
    main            12
    printf          19      25
    putchar         10      36
    putnum          10      35      40
    puts            31      38
    score           16      26
    stdio            7
    string           8
    strlen          24
    tabwidth        17      22      26
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TAIL.C
**  ------------------------------------------------------------------------
**  Display the last n lines of a file (20 lines by default).
**
**  Revision history
**  ================
**  Modified  19930604 by Ruurd Pels:
**  - Increased default line numbers from 5 to 20
**  - Made ANSI C conformant (I hope)
**  - Added '-' support for commandline
**  - Outputs header to stderr instead of stdout to leave it out when
**    redirecting files
**  - Fixed \r\r\n bug for MSDOS machines
**
**  Modified  19861005 by Joe Huffman:
**  - Utilize prototyping, fixed a bug, added (a few) comments and help.
**
**  Written   19860204 by Joe Huffman.
**
**  Not copyrighted.
*/

#include <stdio.h>
#include <stdlib.h>

char            head1[] = {"\n------- \""};
char            head2[] = {" -------\n"};
FILE *          fp;
int             filenum;
int             cc;
unsigned int    linenum = 20;
unsigned int    indx;
long int *      tail;

/*
**  Get the number of lines to display at the "tail" of each file from
**  the command line.
*/

void getlinenum(int n, char * str[])
{
      for (--n; n; --n)
      {
            ++str;
            if ((**str == '/') || (**str == '-'))
            {
                  linenum = atoi(*(str) + 1);
                  if (linenum <= 0)
                        linenum = 20;
            }
      }

      /* Because we save a pointer to the end of the PREVIOUS line */
      linenum++;
}

/*
**  Set the file pointer "fp" to "linenum - 1" lines before the end of
**  the file.
*/

void gettail(void)
{
      unsigned char outstr[15];
      unsigned long int currline = 0L;

      tail = (long int *)malloc(sizeof(*tail) * linenum);
      if (!tail)
      {
            fputs("Insufficient memory.", stderr);
            exit(1);
      }
      tail[0] = ftell(fp);
      indx = 0;

      for (cc = getc(fp); cc != EOF; cc = getc(fp))
      {
            if (cc == '\r')
            {
                  ++currline;
                  cc = getc(fp);
                  if (cc != '\n')
                        ungetc(cc, fp);
                  ++indx;
                  indx %= linenum;
                  tail[indx] = ftell(fp);
            }
            else
            {
                  if (cc == '\n')
                  {
                        ++currline;
                        cc = getc(fp);
                        if (cc != '\r')
                              ungetc(cc, fp);
                        ++indx;
                        indx %= linenum;
```

```
101 |                         tail[indx] = ftell(fp);
102 |                   }
103 |             }
104 |       }
105 |       fputs("\" ", stderr);
106 |       ltoa(currline, (char *)outstr, 10);
107 |       fputs((char *)outstr, stderr);
108 |       fputs(" lines", stderr);
109 |       if (currline >= linenum - 1)
110 |       {
111 |             indx++;
112 |             indx %= linenum;
113 |       }
114 |       else  indx = 0;
115 |
116 |       if (fseek(fp, tail[indx], 0) == -1)
117 |       {
118 |             fputs("\nFile seek error.", stderr);
119 |             exit(1);
120 |       }
121 |       free(tail);
122 | }
123 |
124 | /*
125 | **  Tell the user what the program is and how to use it.
126 | */
127 |
128 | void help(void)
129 | {
130 |       char *ptr;
131 |       static char help_str[] =  "Usage:\n\nTAIL <filename> [filename] "
132 |             "[/n]\n\n<filename>  - The name of a valid file, wildcards "
133 |             "accepted.\nn           - Number of lines to print out, 20 "
134 |             "by default.";
135 |
136 |       for (ptr = &help_str[0]; *ptr; ptr++)
137 |             fputc(*ptr, stdout);
138 | }
139 |
140 | int main(int argc, char **argv)
141 | {
142 |       if (argc <= 1)
143 |       {
144 |             help();
145 |             exit(1);
146 |       }
147 |
148 |       getlinenum(argc, argv);
149 |
150 |       for (filenum = 1; filenum < argc; ++filenum)
151 |       {
152 |             if (*argv[filenum] == '/')
153 |                   continue;
154 |             fp = fopen(argv[filenum], "rb");
155 |             if (!fp)
156 |             {
157 |                   fputs(head1, stderr);
158 |                   fputs(argv[filenum], stderr);
159 |                   fputs("\" not found.", stderr);
160 |                   fputs(head2, stderr);
161 |             }
162 |             else
163 |             {
164 |                   fputs(head1, stderr);
165 |                   fputs(argv[filenum], stderr);
166 |                   gettail();
167 |                   fputs(head2, stderr);
168 |                   for (cc = getc(fp); cc != EOF; cc = getc(fp))
169 |                   {
170 | #ifdef __MSDOS__
171 |                         if (cc != '\r')
172 |                         {
173 |                               fputc(cc, stdout);
174 |                         }
175 | #else
176 |                         fputc(cc, stdout);
177 | #endif
178 |                   }
179 |                   fclose(fp);
180 |             }
181 |       }
182 |       return EXIT_SUCCESS;
183 | }
```

TEXT STATISTICS

```
  4782 characters
   183 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     8 constants [character]
    12 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    EOF              79     168
    EXIT_SUCCESS             182
    FILE             31
    __MSDOS__       170
    argc            140     142     148     150
    argv            140     148     152     154     158     165
    atoi             50
    cc               33      79+     81      84      85      86      93      96      97      98     168+
           171      173     176
    currline         68      83      95     106     109
    exit             74     119     145
    fclose          179
    filenum          32     150+    152     154     158     165
    fopen           154
    fp               31      76      79+     84      86      89      96      98     101     116     154
           155      168+    179
    fputc           137     173     176
    fputs            73     105     107     108     118     157     158     159     160     164     165
           167
    free            121
    fseek           116
    ftell            76      89     101
    getc             79+     84      96     168+
    getlinenum               43     148
    gettail          65     166
    h        26      27
    head1            29     157     164
    head2            30     160     167
    help            128     144
    help_str        131     136
    indx             35      77      87      88      89      99     100     101     111     112     114
           116
    linenum          34      50      51      52      57      70      88     100     109     112
    ltoa            106
    main            140
    malloc           70
    n        43      45+
    outstr           67     106     107
    ptr             130     136+    137
    stderr           73     105     107     108     118     157     158     159     160     164     165
           167
    stdio            26
    stdlib           27
    stdout          137     173     176
    str              43      47      48+     50
    tail             36      70+     71      76      89     101     116     121
    ungetc           86      98
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  SNIPPETS doesn't support many older compilers, but it's easy to add
  5 | **  compiler or version specific header to make things portable. Here's
  6 | **  a sample header for Turbo C 1.5.
  7 | */
  8 |
  9 | #ifndef TC15__H
 10 | #define TC15__H
 11 |
 12 | #include "clock.h"
 13 | #include "strftime.h"
 14 |
 15 | #define signal ssignal
 16 |
 17 | #define FILENAME_MAX 80
 18 |
 19 | #endif /* TC15__H */
```

TEXT STATISTICS

```
   394 characters
    19 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     2 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    FILENAME_MAX            17
    TC15__H          9      10
    signal          15
    ssignal         15
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**
**  This was written under AIX 3.2.5.1
**  (Cripes just a few more numbers please.)
**
**  Terminal functions
**
**  Option 0 - Turns the ECHO on.
**         1 - Turns the ECHO off.
**         2 - Waits forever for keyboard activity.
**
**  Public Domain *IX terminal functions.
**  Slung together by Steve Poole for RBS's SNIPPETS.
**
*/

#define _POSIX_SOURCE 1

#include <termios.h>
#include <sys/types.h>
#include "unxconio.h"

int term_option(option)
int  option;
{
  struct termios attributes;

    switch(option)
    {
/*
**  Turn echo on
*/
      case 0:
             if(tcgetattr(STDIN_FILENO,&attributes) != 0)
               return (-1);
             attributes.c_lflag |= ECHO;
             if(tcsetattr(STDIN_FILENO,TCSANOW,&attributes) != 0)
               return (-1);
             break;
/*
**  Turn echo off
*/
      case 1:
             if(tcgetattr(STDIN_FILENO,&attributes) != 0)
               return (-1);
             attributes.c_lflag &= ~(ECHO);
             if(tcsetattr(STDIN_FILENO,TCSAFLUSH,&attributes) != 0)
               return (-1);
             break;
/*
**  Wait forever for the keyboard to be touched. (AHHHHHH!!!!!!!)
*/
      case 2:
             if(tcgetattr(STDIN_FILENO,&attributes) != 0)
               return (-1);
             attributes.c_lflag    &= ~(ICANON);
             attributes.c_cc[VMIN]  = 1;
             attributes.c_cc[VTIME] = 1;
             if(tcsetattr(STDIN_FILENO,TCSANOW,&attributes) != 0)
               return (-1);
             break;
/*
**  Don't be a bozo, call it with something
*/
      default:
              printf("Error in terminal options routine, "
                     "BAD OPTION %d\n",option);
             return(-1);
             break;
    }
}

#ifdef TEST

main(argc,argv)
int  argc;
char *argv[];
{
  int  option,istat;
  char buffer[2];

  option = atoi(argv[1]);
  istat = term_option(option);
  istat = read(STDIN_FILENO,&buffer,1);
  if(istat < 0)
    printf("Error on READ\n");
  return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  2145 characters
    92 lines

LEXICAL STATISTICS

     7 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     4 constants [string]
    24 constants [numeric]

SYMBOL TABLE

    ECHO            38      48
    ICANON          58
    STDIN_FILENO            36      39      46      49      56      61      86
    TCSAFLUSH       49
    TCSANOW         39      61
    TEST            75
    VMIN            59
    VTIME           60
    _POSIX_SOURCE           19
    argc            77      78
    argv            77      79      84
    atoi            84
    attributes              28      36      38      39      46      48      49      56      58      59
            60      61
    buffer          82      86
    c_cc            59      60
    c_lflag         38      48      58
    h       21      22
    istat           81      85      86      87
    main            77
    option          25      26      30      69      81      84      85
    printf          68      88
    read            86
    sys             22
    tcgetattr       36      46      56
    tcsetattr       39      49      61
    term_option             25      85
    termios         21      28
    types           22
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #ifdef __WATCOMC__
  4 |  #pragma off (unreferenced);
  5 | #endif
  6 |
  7 | /* testcmt.c - test getcmt.exe - C comment 1 */
  8 | int i;
  9 | main()
 10 | {
 11 |     int j;
 12 |     /* C comment 2 */
 13 |     int k;
 14 |     /* C comment 3 */
 15 |     char ch2;    /* C comment 4 */  char ch3;   // C++ comment 1
 16 |     // C++ comment 2
 17 |     char ch4;
 18 |     // C++ comment 3
 19 |     i = 0;
 20 |     return(i);
 21 | }
 22 | /* end of testcmt.c - C comment 5 */
```

TEXT STATISTICS

```
   436 characters
    22 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     3 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     1 constants [numeric]
```

SYMBOL TABLE

```
    __WATCOMC__              3
    ch2             15
    ch3             15
    ch4             17
    i        8      19      20
    j       11
    k       13
    main             9
    off              4
    unreferenced             4
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TEXTMOD.C - Demonstrates techniques for modifying text files
**
**  public domain demo by Bob Stout
*/

#include <string.h>
#include "snipfile.h"

void show_text_file(char *txt)
{
      FILE *in;
      char line[80];

      in = cant("test.txt", "r");
      printf("\n%s:\n\n", txt);

      while (!feof(in))
      {
            if (NULL != fgets(line, 80, in))
                  fputs(line, stdout);
      }
}

void create_text_file(void)
{
      FILE *out;

      out = cant("test.txt", "w");
      fputs("This is a test!\n", out);
      fputs("This is a dummy line to delete...\n", out);
      fputs("This is a dummy line to modify...\n", out);
      fputs("All done!\n", out);
      fclose(out);

      show_text_file("The file as written is");
}

main()
{
      FILE *in, *out;
      char line[80], *tmp, *ptr;

      /*
      **  Open the original file for reading and a temporary file for writing
      */

      create_text_file();
      in  = cant("test.txt", "r");
      tmp = tmpnam(NULL);
      out = cant(tmp, "w");

      /*
      **  Read the first line and copy it
      */

      fgets(line, 80, in);
      fputs(line, out);

      /*
      **  Discard the second line
      */

      fgets(line, 80, in);

      /*
      **  Add a new line
      */

      fputs("(Isn't it?)\n", out);

      /*
      **  Read the 3rd line, modify it, then write it out
      */

      fgets(line, 80, in);
      ptr = strrchr(line, 'm');
      strcpy(ptr, "edit...\n");
      fputs(line, out);

      /*
      **  Read the last line and copy it
      */

      fgets(line, 80, in);
      fputs(line, out);

      /*
      **  Close the files, delete the old, rename the temp
      */

      fclose(in);
      fclose(out);
      remove("test.txt");
      rename(tmp, "test.txt");

      /*
      **  Now let's see the results
```

```
 101 |       */
 102 |
 103 |       show_text_file("The file as modified is");
 104 |
 105 |       return 0;
 106 | }
```

TEXT STATISTICS

```
  2066 characters
   106 lines
```

LEXICAL STATISTICS

```
    10 comments [std-C]
     0 comments [C++]
     2 preprocessor instructions
     1 constants [character]
    19 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    FILE            14      29      43
    NULL            22      52
    cant            17      31      51      53
    create_text_file        27      50
    fclose          36      94      95
    feof            20
    fgets           22      59      66      78      87
    fputs           23      32      33      34      35      60      72      81      88
    h       9
    in              14      17      20      22      43      51      59      66      78      87      94
    line            15      22      23      44      59      60      66      78      79      81      87
           88
    main            41
    out             29      31      32      33      34      35      36      43      53      60      72
           81       88      95
    printf          18
    ptr             44      79      80
    remove          96
    rename          97
    show_text_file          12      38     103
    stdout          23
    strcpy          80
    string           9
    strrchr         79
    tmp             44      52      53      97
    tmpnam          52
    txt             12      18
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TICKTRAP.C - Trap the PC timer interrupt.
**
**  public domain by Bob Stout
*/

#include <dos.h>
#include "ticktrap.h"

static int (*newptr)(void);
static void endtrap(void);

#if defined(__ZTC__) || defined(__SC__)

static int do_new(struct INT_DATA *idp);
static struct INT_DATA idp;

/*
**  Installs a user-defined funtion in an Int 1Ch ISR. The user-supplied
**  function should take no arguments and should return an int. If zero
**  is returned from the user-supplied function, the previous Int 1Ch ISR
**  is called upon exit. If non-zero is returned from the user-supplied
**  function, the previous ISR is not called.
**
**  Side effect: Registers endtrap() with atexit() to restore Int 1Ch
**               upon program termination.
*/

void ticktrap(int (*fptr)(void))
{
      newptr = fptr;
      int_intercept(0x1c, do_new, 0);
      atexit(endtrap);
}

/*
**  Calls the previous Int 1Ch ISR
*/

void tickchain(void)
{
      int_prev(&idp);
}

static int do_new(struct INT_DATA *idp)
{
      return (*newptr)();
}

static void endtrap(void)
{
      int_restore(0x1c);
}

#else /* It's not Symantec or Zortech */

static void (INTERRUPT FAR *old1c)(void);

static void INTERRUPT FAR do_new(void);

/*
**  Installs a user-defined funtion in an Int 1Ch ISR. The user-supplied
**  function should take no arguments and should return an int. If zero
**  is returned from the user-supplied function, the previous Int 1Ch ISR
**  is called upon exit. If non-zero is returned from the user-supplied
**  function, the previous ISR is not called.
**
**  Side effect: Registers endtrap() with atexit() to restore Int 1Ch
**               upon program termination.
*/

void ticktrap(int (*fptr)(void))
{
      old1c = _dos_getvect(0x1c);
      newptr = fptr;
      disable();
      _dos_setvect(0x1c, do_new);
      enable();
      atexit(endtrap);
}

/*
**  Calls the previous Int 1Ch ISR
*/

void tickchain(void)
{
      (*old1c)();
}

static void INTERRUPT FAR do_new(void)
{
      int retval;
      retval = (*newptr)();
      if (0 == retval)
            (*old1c)();
}
```

```
 101 | static void endtrap(void)
 102 | {
 103 |       disable();
 104 |       _dos_setvect(0x1c, old1c);
 105 |       enable();
 106 | }
 107 |
 108 | #endif
```

TEXT STATISTICS

```
  2337 characters
   108 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     1 constants [string]
     7 constants [numeric]
```

SYMBOL TABLE

```
    FAR             59      61      93
    INTERRUPT       59      61      93
    INT_DATA        17      18      47
    __SC__          15
    __ZTC__         15
    _dos_getvect            76
    _dos_setvect            79     104
    atexit          35      81
    defined         15+
    disable         78     103
    do_new          17      34      47      61      79      93
    dos              9
    enable          80     105
    endtrap         13      35      52      81     101
    fptr            31      33      74      77
    h        9
    idp             17      18      44      47
    int_intercept           34
    int_prev        44
    int_restore             54
    newptr          12      33      49      77      96
    old1c           59      76      90      98     104
    retval          95      96      97
    tickchain       42      88
    ticktrap        31      74
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  TICKTRAP.H - Trap the PC timer interrupt.
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #ifndef TICKTRAP__H
 10 | #define TICKTRAP__H
 11 |
 12 | #include <stdlib.h>
 13 | #include "extkword.h"
 14 |
 15 | #ifdef __IS_ZORTECH__
 16 |  #include <int.h>
 17 | #else
 18 |  #include "pchwio.h"
 19 | #endif
 20 |
 21 | void ticktrap(int (*fptr)(void));
 22 | void tickchain(void);
 23 |
 24 | #endif /* TICKTRAP__H */
```

TEXT STATISTICS

```
   394 characters
    24 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     0 constants [character]
     2 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    TICKTRAP__H              9      10
    __IS_ZORTECH__          15
    fptr            21
    h       12      16
    stdlib          12
    tickchain       22
    ticktrap        21
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  TIMEGETC.C - waits for a given number of seconds for the user to press
 5 | **               a key.  Returns the key pressed, or EOF if time expires
 6 | **
 7 | **  by Bob Jarvis
 8 | */
 9 |
10 | #include <stdio.h>
11 | #include <time.h>
12 | #include <conio.h>
13 | #include "snipkbio.h"
14 |
15 |
16 | int timed_getch(int n_seconds)
17 | {
18 |       time_t start, now;
19 |
20 |       start = time(NULL);
21 |       now = start;
22 |
23 |       while(difftime(now, start) < (double)n_seconds && !kbhit())
24 |       {
25 |             now = time(NULL);
26 |       }
27 |
28 |       if(kbhit())
29 |             return getch();
30 |       else  return EOF;
31 | }
32 |
33 | #ifdef TEST
34 |
35 | main()
36 | {
37 |       int c;
38 |
39 |       printf("Starting a 5 second delay...\n");
40 |
41 |       c = timed_getch(5);
42 |
43 |       if(c == EOF)
44 |             printf("Timer expired\n");
45 |       else  printf("Key was pressed, c = '%c'\n", c);
46 |       return 0;
47 | }
48 |
49 | #endif /* TEST */
```

```
TEXT STATISTICS

   899 characters
    49 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     4 constants [string]
     2 constants [numeric]

SYMBOL TABLE

    EOF             30      43
    NULL            20      25
    TEST            33
    c       37      41      43      45
    conio           12
    difftime        23
    getch           29
    h       10      11      12
    kbhit           23      28
    main            35
    n_seconds       16      23
    now             18      21      23      25
    printf          39      44      45
    start           18      20      21      23
    stdio           10
    time            11      20      25
    time_t          18
    timed_getch             16      41
```

```
/* +++Date last modified: 05-Jul-1997 */

/*_ timer.c   Mon Feb 29 1988   Modified by: Jerry Coffin   */
/* Written by Walter Bright                                 */
/* To compile (with Zortech C):                             */
/*    ZTC -mti timer                                        */

#include    <stdio.h>
#include    <time.h>
#include    <process.h>
#include    <stdlib.h>
#include    <string.h>
#include    "extkword.h"

#if defined(_QC) || defined(_MSC_VER)
 #include    <malloc.h>
#endif

#ifdef __SC__
 #define SVP_CAST(x) (const char * const *)(x)
#else
 #define SVP_CAST(x) (x)
#endif

#ifdef __ZTC__
int _okbigbuf = 0;      /* Use as little memory as possible */
#endif

main(int argc, char *argv[])
{
      clock_t clock(),starttime;
      int status, i;

      if (argc < 2)
      {     printf("Time execution of a command.\nUse:\n\ttimer command\n");
            exit(1);
      }
      argv[argc] = 0;         /* terminate with a 0 (unportable method) */
      starttime = clock();

/*
** This block added to keep arguments with embedded whitespace from getting
** broken up when timing is done.  A *nix version should be a bit different.
** Also, the assignment to argv[i] technically isn't portable, but I don't
** know of an implementation where it causes a problem. <JC>
*/
      for (i=1;i<argc;i++)
      {
            if (NULL != strchr(argv[i], ' '))
            {
                  size_t len = strlen(argv[i]+3);
                  char *temp = malloc(len);

                  strcpy(temp+1, argv[i]);
                  temp[0] = '"';
                  temp[len-2] = '"';
                  temp[len-1] = '\0';
                  argv[i] = temp;
            }
      }

/* This reduces memory usage with MS compilers by releasing unused heap.
 * Watcom may support it as well, but I'm not sure.
 */
#if defined(_QC) || defined(_MSC_VER)
      _heapmin();
#endif
      status = spawnvp(0,argv[1],SVP_CAST(argv + 1));
      starttime = clock() - starttime;
      if (status == -1)
      {     printf("'%s' failed to execute\n",argv[1]);
            exit(1);
      }
#if !defined(__TURBOC__)
      printf("Elapsed time = %d.%02d seconds\n",(int) (starttime/CLK_TCK),
            (int) (starttime%CLK_TCK));
#else
      printf("Elapsed time = %.2f seconds\n", starttime/CLK_TCK);
#endif
      if (status != 0)
            printf("--- errorlevel %d\n",status);
      /* exit(0); changed to `return 0;' to reduce program size (minutely)
       * and possibility or warnings that main doesn't return a value.
       */
      return 0;
}

#ifdef __ZTC__

/* Prevent exit() and fclose() from being linked in from library  */
/* (to conserve the size of the output file).                     */

void exit(int exitstatus)
{
      _exit(exitstatus);
}

#endif
```

TEXT STATISTICS

```
  2875 characters
    98 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    23 preprocessor instructions
     4 constants [character]
     6 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    CLK_TCK          75      76      78
    NULL             49
    SVP_CAST         20      22      68
    _MSC_VER         15      65
    _QC              15      65
    __SC__           19
    __TURBOC__               74
    __ZTC__          25      88
    _exit            95
    _heapmin         66
    _okbigbuf        26
    argc             29      34      38      47
    argv             29      38      49      51      54      58      68+     71
    clock            31      39      69
    clock_t          31
    defined          15+     65+     74
    exit             36      72      93
    exitstatus               93      95
    h         8       9      10      11      12      16
    i        32      47+     49      51      54      58
    len              51      52      56      57
    main             29
    malloc           16      52
    printf           35      71      75      78      81
    process          10
    size_t           51
    spawnvp          68
    starttime        31      39      69+     75      76      78
    status           32      68      70      80      81
    stdio             8
    stdlib           11
    strchr           49
    strcpy           54
    string           12
    strlen           51
    temp             52      54      55      56      57      58
    time              9
    x        20+     22+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 /-------------------------------------\
|     TIME_MAC_CONV                    |-----------------------------------|
|\-------------------------------------/
|
| This routine converts the macro  __TIME__ to standard format
|
| Example:   "11:50:21" -> "115021"
|
|-------------------------------------------------------------------------|
| CALL:
|    strcpy(string, time_mac_conv(string) );
|
| HEADER:
|    string.h    : strcpy
|    stdio.h     : sscanf, sprintf
|
| GLOBALE VARIABLES:
|    %
|
| ARGUMENTS:
|    pszTime      : String with time in  __TIME__ format (HH:MM:SS)
|
| PROTOTYPE:
|    char far *time_mac_conv(char *pszTime);
|
| RETURN VALUE:
|    char szStr   : Time in format HHMMSS
|
| MODULE:
|    time__.c
|-------------------------------------------------------------------------|
|
|
|-------------------------------------------------------------------------|
|1992-09-16/Erik Bachmann
\-------------------------------------------------------------------------|*/

char _CfnTYPE *time_mac_conv(char *pszTime)
{
      char  szStr[12];                              /* Convertion string */
      char  hh[3],                                  /* Hour     */
            mm[3],                                  /* Minutes  */
            ss[3];                                  /* Seconds  */

      /*---------------------------------------------*/

      strcpy(szStr, pszTime);                       /* Copy string */

      sscanf(szStr, "%2s %*c %2s %*c %2s", hh, mm, ss);

      /* Split the string into basics */

      sprintf(szStr, "%s%s%s", hh,mm,ss);

      /* Assemble new string */

      return((char *) szStr);                       /* Return new string */
}
```

```
TEXT STATISTICS

  1823 characters
    61 lines

LEXICAL STATISTICS

    11 comments [std-C]
     0 comments [C++]
     0 preprocessor instructions
     0 constants [character]
     2 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    _CfnTYPE        41
    hh              44      52      56
    mm              45      52      56
    pszTime         41      50
    sprintf         56
    ss              46      52      56
    sscanf          52
    strcpy          50
    szStr           43      50      52      56      60
    time_mac_conv           41
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 | **  To4Dos.C - Utility to convert SNIPPETS.NDX to 4DOS DESCRIPT.ION file.
 5 | */
 6 |
 7 | #include <stdlib.h>
 8 | #include <string.h>
 9 | #include <ctype.h>
10 | #include "snipfile.h"
11 |
12 | main()
13 | {
14 |       FILE *ndx, *desc;
15 |       char line1[81], line2[81], *ptr;
16 |       int i;
17 |
18 |       ndx  = cant("snippets.ndx", "r");
19 |       desc = cant("descript.ion", "w");
20 |
21 |       while (!feof(ndx))
22 |       {
23 |             if (NULL != (fgets(line1, 81, ndx)))
24 |             {
25 |                   if ('|' == *line1 || 3 > strlen(line1) ||
26 |                         isspace(line1[2]) || '-' == *line1)
27 |                   {
28 |                         continue;
29 |                   }
30 |                   for (ptr = line1 + 2; ' ' != *ptr; ++ptr)
31 |                         fputc(*ptr, desc);
32 |                   fputs(" <", desc);
33 |                   if (isspace(line1[19]))
34 |                   {
35 |                         for (i = 15; i < 18; ++i)
36 |                               fputc(line1[i], desc);
37 |                         fprintf(desc, ">%s", line1 + 19);
38 |                   }
39 |                   else
40 |                   {
41 |                         for (i = 15; i < 18; ++i)
42 |                               fputc(line2[i], desc);
43 |                         fprintf(desc, ">%s", line2 + 19);
44 |                   }
45 |                   strcpy(line2, line1);
46 |             }
47 |       }
48 |       return EXIT_SUCCESS;
49 | }
```

TEXT STATISTICS

```
  1409 characters
    49 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     3 constants [character]
     8 constants [string]
    13 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_SUCCESS            48
    FILE            14
    NULL            23
    cant            18      19
    ctype            9
    desc            14      19      31      32      36      37      42      43
    feof            21
    fgets           23
    fprintf         37      43
    fputc           31      36      42
    fputs           32
    h        7       8       9
    i       16      35+     36      41+     42
    isspace         26      33
    line1           15      23      25+     26+     30      33      36      37      45
    line2           15      42      43      45
    main            12
    ndx             14      18      21      23
    ptr             15      30+     31
    stdlib           7
    strcpy          45
    string           8
    strlen          25
```

```
/* +++Date last modified: 05-Jul-1997 */

#include "a2e.h"

int ascii2ebcdic[256] = {
          0,  1,  2,  3, 55, 45, 46, 47, 22,  5, 37, 11, 12, 13, 14, 15,
         16, 17, 18, 19, 60, 61, 50, 38, 24, 25, 63, 39, 28, 29, 30, 31,
         64, 79,127,123, 91,108, 80,125, 77, 93, 92, 78,107, 96, 75, 97,
        240,241,242,243,244,245,246,247,248,249,122, 94, 76,126,110,111,
        124,193,194,195,196,197,198,199,200,201,209,210,211,212,213,214,
        215,216,217,226,227,228,229,230,231,232,233, 74,224, 90, 95,109,
        121,129,130,131,132,133,134,135,136,137,145,146,147,148,149,150,
        151,152,153,162,163,164,165,166,167,168,169,192,106,208,161,  7,
         32, 33, 34, 35, 36, 21,  6, 23, 40, 41, 42, 43, 44,  9, 10, 27,
         48, 49, 26, 51, 52, 53, 54,  8, 56, 57, 58, 59,  4, 20, 62,225,
         65, 66, 67, 68, 69, 70, 71, 72, 73, 81, 82, 83, 84, 85, 86, 87,
         88, 89, 98, 99,100,101,102,103,104,105,112,113,114,115,116,117,
        118,119,120,128,138,139,140,141,142,143,144,154,155,156,157,158,
        159,160,170,171,172,173,174,175,176,177,178,179,180,181,182,183,
        184,185,186,187,188,189,190,191,202,203,204,205,206,207,218,219,
        220,221,222,223,234,235,236,237,238,239,250,251,252,253,254,255
};

int ebcdic2ascii[256] = {
          0,  1,  2,  3,156,  9,134,127,151,141,142, 11, 12, 13, 14, 15,
         16, 17, 18, 19,157,133,  8,135, 24, 25,146,143, 28, 29, 30, 31,
        128,129,130,131,132, 10, 23, 27,136,137,138,139,140,  5,  6,  7,
        144,145, 22,147,148,149,150,  4,152,153,154,155, 20, 21,158, 26,
         32,160,161,162,163,164,165,166,167,168, 91, 46, 60, 40, 43, 33,
         38,169,170,171,172,173,174,175,176,177, 93, 36, 42, 41, 59, 94,
         45, 47,178,179,180,181,182,183,184,185,124, 44, 37, 95, 62, 63,
        186,187,188,189,190,191,192,193,194, 96, 58, 35, 64, 39, 61, 34,
        195, 97, 98, 99,100,101,102,103,104,105,196,197,198,199,200,201,
        202,106,107,108,109,110,111,112,113,114,203,204,205,206,207,208,
        209,126,115,116,117,118,119,120,121,122,210,211,212,213,214,215,
        216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,
        123, 65, 66, 67, 68, 69, 70, 71, 72, 73,232,233,234,235,236,237,
        125, 74, 75, 76, 77, 78, 79, 80, 81, 82,238,239,240,241,242,243,
         92,159, 83, 84, 85, 86, 87, 88, 89, 90,244,245,246,247,248,249,
         48, 49, 50, 51, 52, 53, 54, 55, 56, 57,250,251,252,253,254,255
};
```

TEXT STATISTICS

```
  2494 characters
    41 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     1 constants [string]
   514 constants [numeric]
```

SYMBOL TABLE

```
    ascii2ebcdic             5
    ebcdic2ascii            24
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  TODAY.C - Return today's date in a scalar date format.
  5 | **
  6 | **  public domain by Bob Stout - uses Ray Gardner's SCALDATE.C
  7 | */
  8 |
  9 | #include <time.h>
 10 | #include "scaldate.h"
 11 |
 12 |
 13 | long today(void)
 14 | {
 15 |       time_t tnow;
 16 |       struct tm *tmnow;
 17 |
 18 |       time(&tnow);
 19 |       tmnow = localtime(&tnow);
 20 |       return ymd_to_scalar(tmnow->tm_year + 1900, tmnow->tm_mon + 1,
 21 |             tmnow->tm_mday);
 22 | }
 23 |
 24 | #ifdef TEST
 25 |
 26 | #include <stdio.h>
 27 |
 28 | main()
 29 | {
 30 |       unsigned yr, mo, dy;
 31 |       long dnow = today();
 32 |
 33 |       scalar_to_ymd(dnow, &yr, &mo, &dy);
 34 |       printf("Today is %d/%d/%d\n", mo, dy, yr);
 35 |       return 0;
 36 | }
 37 |
 38 | #endif /*TEST */
```

TEXT STATISTICS

```
   693 characters
    38 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     2 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    TEST            24
    dnow            31      33
    dy              30      33      34
    h        9      26
    localtime       19
    main            28
    mo              30      33      34
    printf          34
    scalar_to_ymd           33
    stdio           26
    time             9      18
    time_t          15
    tm              16
    tm_mday         21
    tm_mon          20
    tm_year         20
    tmnow           16      19      20+     21
    tnow            15      18      19
    today           13      31
    ymd_to_scalar           20
    yr              30      33      34
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TODAYBAK.C - Back up today's work to a specified floppy
**
**  public domain demo by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include "sniptype.h"
#include "cast.h"
#include "dirport.h"

#if defined(__ZTC__) && (__ZTC__ < 0x600)
 #define _dos_getdate   dos_getdate
 #define _dos_setdate   dos_setdate
 #define _dosdate_t     dos_date_t
#endif
#ifdef __TURBOC__
 #define _dosdate_t     dosdate_t
#endif

struct DOS_DATE {
        unsigned int da : 5;
        unsigned int mo : 4;
        unsigned int yr : 7;
        } ;

struct _dosdate_t today;
struct DOS_DATE   ftoday;
char   drive;

void do_dir(char *);
void usage(void);

int main(int argc, char *argv[])
{
      int i;

      _dos_getdate(&today);
      ftoday.da = today.day;
      ftoday.mo = today.month;
      ftoday.yr = today.year - 1980;

      if (2 > argc)
            usage();

      drive = *argv[1];
      if (!strchr("AaBb", drive))
            usage();

      if (3 > argc)
            do_dir(".");
      else for (i = 2; i < argc; ++i)
            do_dir(argv[i]);

      return EXIT_SUCCESS;
}

void usage(void)
{
      puts("usage: TODAYBAK floppy [dir1] [...dirN]");
      puts("       Copies today's files to the specified floppy.");
      puts("       floppy = 'A' | 'B'");
      puts("       with no directories specified, "
            "uses current directory");
      exit(EXIT_FAILURE);
}

void do_dir(char *path)
{
      char search[67];
      DOSFileData ff;

      strcat(strcpy(search, path), "\\*.*");
      if (Success_ == FIND_FIRST(search, 0xff, &ff)) do
      {
            if (!(ff_attr(&ff) & _A_SUBDIR) && '.' != *(ff_name(&ff)))
            {
                  if (CAST(unsigned short, ff_date(&ff)) ==
                        CAST(unsigned short, ftoday))
                  {
                        char cmd[128];

                        sprintf(cmd, "COPY %s\\%s %c: > NUL",
                              path, ff_name(&ff), drive);
                        system(cmd);
                  }
            }
      } while (Success_ == FIND_NEXT(&ff));
      FIND_END(&ff);
}
```

TEXT STATISTICS

2263 characters
95 lines

LEXICAL STATISTICS

2 comments [std-C]
0 comments [C++]
15 preprocessor instructions
1 constants [character]
12 constants [string]
12 constants [numeric]

SYMBOL TABLE

CAST	83	84					
DOSFileData		76					
DOS_DATE	26	33					
EXIT_FAILURE		70					
EXIT_SUCCESS		60					
FIND_END	94						
FIND_FIRST		79					
FIND_NEXT	93						
Success_	79	93					
_A_SUBDIR	81						
__TURBOC__		22					
__ZTC__	17+						
_dos_getdate		18	43				
_dos_setdate		19					
_dosdate_t		20	23	32			
argc	39	48	55	57			
argv	39	51	58				
cmd	86	88	90				
da	27	44					
day	44						
defined	17						
do_dir	36	56	58	73			
dos	12						
dos_date_t		20					
dos_getdate		18					
dos_setdate		19					
dosdate_t	23						
drive	34	51	52	89			
exit	70						
ff	76	79	81+	83	89	93	94
ff_attr	81						
ff_date	83						
ff_name	81	89					
ftoday	33	44	45	46	84		
h 9	10	11	12				
i 41	57+	58					
main	39						
mo	28	45					
month	45						
path	73	78	89				
puts	65	66	67	68			
search	75	78	79				
sprintf	88						
stdio	9						
stdlib	10						
strcat	78						
strchr	52						
strcpy	78						
string	11						
system	90						
today	32	43	44	45	46		
usage	37	49	53	63			
year	46						
yr	29	46					

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  This is a copyrighted work which is functionally identical to work
**  originally published in Micro Cornucopia magazine (issue #52, March-April,
**  1990) and is freely licensed by the author, Walter Bright, for any use.
*/

/*_ toolkit.h   Tue Apr 18 1989   Modified by: Walter Bright */

#ifndef TOOLKIT_H
#define TOOLKIT_H

/* Define stuff that's different between machines.
 * PROTOTYPING          1 if compiler supports prototyping
 * HOSTBYTESWAPPED      1 if on the host machine the bytes are
 *                swapped (1 for 6809, 68000, 0 for 8088
 *                and VAX).
 */

#if defined(MSDOS) || defined(__MSDOS__)
#define PROTOTYPING     1
#define HOSTBYTESWAPPED 0

#define BITSPERBYTE 8
#define SIZEOFINT   sizeof(int)
#define SIZEOFLONG  sizeof(long)

#else
#ifdef M_UNIX     /* SCO UNIX using Microsoft C. */
#define PROTOTYPING     1
#define HOSTBYTESWAPPED 0
#define EXIT_SUCCESS    0
#define EXIT_FAILURE    1

#define BITSPERBYTE 8
#define SIZEOFINT   sizeof(int)
#define SIZEOFLONG  sizeof(long)
#else                   /* NOTE: host.h is *NOT* included in SNIPPETS    */
#include    "host.h"    /* Compiler/environment-specific stuff goes here */
#endif

#endif

/* Static definitions do not appear in the linker .MAP file. Override   */
/* the definition here to make them global if necessary.          */
#ifndef STATIC
#define STATIC    static
#endif

#define arraysize(array)      (sizeof(array) / sizeof(array[0]))

/* Macros so that we can do prototyping, but still work with non- */
/* prototyping compilers:                                         */

#if PROTOTYPING
#define P(s)      s
#else
#define P(s)      ()
#endif

#ifdef DEBUG
#define debug(a)  (a)
#else
#define debug(a)
#endif

#endif /* TOOLKIT_H */
```

```
TEXT STATISTICS

  1859 characters
    68 lines

LEXICAL STATISTICS

    12 comments [std-C]
     0 comments [C++]
    36 preprocessor instructions
     0 constants [character]
     1 constants [string]
     9 constants [numeric]

SYMBOL TABLE

    BITSPERBYTE             25      36
    DEBUG           62
    EXIT_FAILURE            34
    EXIT_SUCCESS            33
    HOSTBYTESWAPPED         23      32
    MSDOS           21
    M_UNIX          30
    P       57      59
    PROTOTYPING             22      31      56
    SIZEOFINT       26      37
    SIZEOFLONG              27      38
    STATIC          47      48
    TOOLKIT_H       11      12
    __MSDOS__       21
    a       63+     65
    array           51+
    arraysize       51
    debug           63      65
    defined         21+
    s       57+     59
```

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*--------------------------------------------------------------------------*
 4 | * Program:     touch                                                       *
 5 | * Programmer:  Ray L. McVay                                                *
 6 | * Started:     8 Aug 91                                                    *
 7 | * Updated:     13 Feb 93 Thad Smith                                        *
 8 | * Updated:     15 Feb 93 Bob Stout                                         *
 9 | *--------------------------------------------------------------------------*
10 | * Simple touch program to test BC time stamping function.                  *
11 | * Public Domain                                                            *
12 | *--------------------------------------------------------------------------*/
13 |
14 | #include <stdio.h>
15 | #include <stdlib.h>
16 | #include <time.h>
17 | #if defined(__TURBOC__) || defined(__SC__)
18 |  #include <dos.h>
19 |  #include <io.h>
20 | #else
21 |  #include "ftime.h"                /* Borland work-alike in SNIPPETS   */
22 | #endif
23 |
24 | void usage(void);
25 |
26 | main(int argc, char **argv)
27 | {
28 |       time_t    tnow;
29 |       struct tm tmnow;
30 |       struct ftime ft;
31 |       FILE *f;
32 |
33 |       if (argc < 2)
34 |             usage();
35 |
36 |       tnow  = time(NULL);
37 |       tmnow = *localtime(&tnow);
38 |
39 |       ft.ft_year  = tmnow.tm_year - 80;
40 |       ft.ft_month = tmnow.tm_mon + 1;
41 |       ft.ft_day   = tmnow.tm_mday;
42 |       ft.ft_hour  = tmnow.tm_hour;
43 |       ft.ft_min   = tmnow.tm_min;
44 |       ft.ft_tsec  = tmnow.tm_sec/2;
45 |
46 |       if ((f = fopen(argv[1], "r+b")) != NULL)
47 |             setftime(fileno(f), &ft);
48 |       else if ((f = fopen(argv[1], "w")) != NULL)
49 |             setftime(fileno(f), &ft);
50 |       else  perror("Can't open file");
51 |
52 |       if (f)
53 |             fclose(f);
54 |
55 |       return EXIT_SUCCESS;
56 | }
57 |
58 | void usage(void)
59 | {
60 |       puts("Usage: TOUCH filename\n");
61 |       puts("  The timestamp of filename will be set to the current time.");
62 |       puts("  A zero-length file will be created if the file doesn't exist.");
63 |       exit(EXIT_FAILURE);
64 | }
```

TEXT STATISTICS

2046 characters
64 lines

LEXICAL STATISTICS

3 comments [std-C]
0 comments [C++]
9 preprocessor instructions
0 constants [character]
7 constants [string]
6 constants [numeric]

SYMBOL TABLE

EXIT_FAILURE		63							
EXIT_SUCCESS		55							
FILE	31								
NULL	36	46	48						
__SC__	17								
__TURBOC__		17							
argc	26	33							
argv	26	46	48						
defined	17+								
dos	18								
exit	63								
f	31	46	47	48	49	52	53		
fclose	53								
fileno	47	49							
fopen	46	48							
ft	30	39	40	41	42	43	44	47	49
ft_day	41								
ft_hour	42								
ft_min	43								
ft_month	40								
ft_tsec	44								
ft_year	39								
ftime	30								
h	14	15	16	18	19				
io	19								
localtime	37								
main	26								
perror	50								
puts	60	61	62						
setftime	47	49							
stdio	14								
stdlib	15								
time	16	36							
time_t	28								
tm	29								
tm_hour	42								
tm_mday	41								
tm_min	43								
tm_mon	40								
tm_sec	44								
tm_year	39								
tmnow	29	37	39	40	41	42	43	44	
tnow	28	36	37						
usage	24	34	58						

```
/* +++Date last modified: 05-Jul-1997 */

/*
** Public Domain by Jerry Coffin.
**
** Interprets a string in a manner similar to the way the compiler
** does string literals in a program.  All escape sequences are
** longer than their translated equivalent, so the string is
** translated in place and either remains the same length or
** becomes shorter.
*/

#include <string.h>
#include <stdio.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *translate(char *string)
{
      char *here=string;
      size_t len=strlen(string);
      int num;
      int numlen;

      while (NULL!=(here=strchr(here,'\\')))
      {
            numlen=1;
            switch (here[1])
            {
            case '\\':
                  break;

            case 'r':
                  *here = '\r';
                  break;

            case 'n':
                  *here = '\n';
                  break;

            case 't':
                  *here = '\t';
                  break;

            case 'v':
                  *here = '\v';
                  break;

            case 'a':
                  *here = '\a';
                  break;

            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
                  numlen = sscanf(here,"%o",&num);
                  *here = (char)num;
                  break;

            case 'x':
                  numlen = sscanf(here,"%x",&num);
                  *here = (char) num;
                  break;
            }
            num = here - string + numlen;
            here++;
            memmove(here,here+numlen,len-num );
      }
      return string;
}

#if defined(__cplusplus) && __cplusplus
 }
#endif
```

TEXT STATISTICS

```
  1888 characters
    82 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
    21 constants [character]
     4 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    NULL            28
    __cplusplus             17+     80+
    defined         17      80
    h       13      14
    here            23      28+     31      37      41      45      49      53      64      65      69
            70      73      74      75+
    len             24      75
    memmove         75
    num             25      64      65      69      70      73      75
    numlen          26      30      64      69      73      75
    size_t          24
    sscanf          64      69
    stdio           14
    strchr          28
    string          13      21      23      24      73      77
    strlen          24
    translate       21
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Demonstrate TRAPFLAG.ASM
**
**  public domain by Bob Stout
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include "extkword.h"

#ifdef __WATCOMC__
 #pragma aux ins09  "_*" parm caller [] modify [ax bx cx dx es]
 #pragma aux undo09 "_*" parm caller [] modify [ax bx cx dx es]
#endif

extern void ins09(void);
extern void undo09(void);

extern volatile int FAR ccrcvd;

static void biosprt(char *p)
{
      union REGS regs;

      while (*p)
      {
            regs.h.ah = 0x0e;             /* Low-level services only!     */
            regs.h.al = *p++;
            regs.x.bx = 0;
            int86(0x10, &regs, &regs);
      }
}

static void FAR my_cc(void)
{
      char *p1 = "Ctrl-";
      char *p2 = "C";
      char *p3 = "Break";
      char *p4 = " received\r\n";

      biosprt(p1);
      if (1 == ccrcvd)
            biosprt(p2);
      else  biosprt(p3);
      biosprt(p4);
}

main()
{
      int ch = 0;

      setbuf(stdout, NULL);
      my_cc();
      ins09();
      atexit(undo09);
      puts("New Ints 09h & 1Bh installed...");
      puts("Hit Esc to quit...");
      do
      {
            if (kbhit())
            {
                  if (0x1b != (ch = getch()))
                  {
                        if (0x20 > ch)
                        {
                              fputc('^', stdout);
                              ch += '@';
                        }
                        fputc(ch, stdout);
                  }
            }
            if (ccrcvd)
            {
                  my_cc();
                  ccrcvd = 0;
            }
      } while (0x1b != ch);
      return 0;
}
```

```
TEXT STATISTICS

  1775 characters
    83 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     2 constants [character]
     9 constants [string]
    10 constants [numeric]

SYMBOL TABLE

    FAR             23      38
    NULL            56
    REGS            27
    __WATCOMC__             15
    ah              31
    al              32
    atexit          59
    aux             16      17
    ax              16      17
    biosprt         25      45      47      48      49
    bx              16      17      33
    caller          16      17
    ccrcvd          23      46      76      79
    ch              54      66      68      71      73      81
    conio           12
    cx              16      17
    dos             11
    dx              16      17
    es              16      17
    fputc           70      73
    getch           66
    h        9      10      11      12      31      32
    ins09           16      20      58
    int86           34
    kbhit           64
    main            52
    modify          16      17
    my_cc           38      57      78
    p       25      29      32
    p1              40      45
    p2              41      47
    p3              42      48
    p4              43      49
    parm            16      17
    puts            60      61
    regs            27      31      32      33      34+
    setbuf          56
    stdio            9
    stdlib          10
    stdout          56      70      73
    undo09          17      21      59
    x       33
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  TREEDIR.C - simple recursive directory lister
  5 | **
  6 | **  public domain demo by Bob Stout
  7 | */
  8 |
  9 | #include <stdio.h>
 10 | #include <string.h>
 11 | #include "sniptype.h"
 12 | #include "dirport.h"
 13 |
 14 | void do_dir(char *path)
 15 | {
 16 |       char search[67], newpath[67];
 17 |       DOSFileData ff;
 18 |
 19 |       strcat(strcpy(search, path), "\\*.*");
 20 |       if (Success_ == FIND_FIRST(search, _A_ANY, &ff)) do
 21 |       {
 22 |             printf("%s\\%s\n", path, ff_name(&ff));
 23 |             if (ff_attr(&ff) & _A_SUBDIR && '.' != *ff_name(&ff))
 24 |             {
 25 |                   strcat(strcat(strcpy(newpath, path), "\\"), ff_name(&ff));
 26 |                   do_dir(newpath);
 27 |             }
 28 |       } while (Success_ == FIND_NEXT(&ff));
 29 |       FIND_END(&ff);
 30 | }
 31 |
 32 | main()                  /* simple recursive current directory lister    */
 33 | {
 34 |       do_dir(".");
 35 |       return 0;
 36 | }
```

TEXT STATISTICS

```
   893 characters
    36 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     1 constants [character]
     6 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             17
    FIND_END        29
    FIND_FIRST              20
    FIND_NEXT       28
    Success_        20      28
    _A_ANY          20
    _A_SUBDIR       23
    do_dir          14      26      34
    ff              17      20      22      23+     25      28      29
    ff_attr         23
    ff_name         22      23      25
    h       9       10
    main            32
    newpath         16      25      26
    path            14      19      22      25
    printf          22
    search          16      19      20
    stdio            9
    strcat          19      25+
    strcpy          19      25
    string          10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*****************************************************/
  4 | /* Filespec  :  triglib.c triglib.h                  */
  5 | /* Date      :  February 21 1997                     */
  6 | /* Time      :  14:11                                */
  7 | /* Revision  :  1.0C                                 */
  8 | /* Update    :                                       */
  9 | /*****************************************************/
 10 | /* Programmer:  Nigel Traves                         */
 11 | /* Address   :  5 Breamer Road, Collingham, Newark,  */
 12 | /*           :  Notts, U.K.                          */
 13 | /* Post Code :  NG23 7PN                             */
 14 | /*****************************************************/
 15 | /* Released to the Public Domain                     */
 16 | /*****************************************************/
 17 |
 18 | #include <math.h>
 19 | #include "triglib.h"
 20 |
 21 | #undef  PI
 22 | #define PI              3.14159265358979323846
 23 | #define PI_Times2       6.28318530717958647692        /* PI * 2 */
 24 | #define PI_Divided_By2  1.57079632679489661923;       /* PI / 2 */
 25 |
 26 | double asinh(double x)
 27 | {
 28 |       return (log(x + sqrt(x * x + 1.0)));
 29 | }
 30 |
 31 | double acosh(double x)
 32 | {
 33 |       return (log(x + sqrt(x * x - 1.0)));
 34 | }
 35 |
 36 | double atanh(double x)
 37 | {
 38 |       return (log((1.0 + x) / (1.0 - x)) / 2.0);
 39 | }
 40 |
 41 | void normalize_radians(double *radians)
 42 | {
 43 |       while (*radians > PI)
 44 |             *radians -= PI_Times2;
 45 |       while (*radians < -PI)
 46 |             *radians += PI_Times2;
 47 | }
 48 |
 49 | void normalize_degrees(double *degrees)
 50 | {
 51 |       while (*degrees > 180.0)
 52 |             *degrees -= 360.0;
 53 |       while (*degrees < -180.0)
 54 |             *degrees += 360.0;
 55 | }
 56 |
 57 | void normalize_grade(double *grade)
 58 | {
 59 |       while (*grade > 200.0)
 60 |             *grade -= 400.0;
 61 |       while (*grade < -200.0)
 62 |             *grade += 400.0;
 63 | }
 64 |
 65 | double radians_to_degrees(double radians)
 66 | {
 67 |       normalize_radians(&radians);
 68 |       return (radians * (180.0 / PI));
 69 | }
 70 |
 71 | double radians_to_grade(double radians)
 72 | {
 73 |       normalize_radians(&radians);
 74 |       return (radians * (200.0 / PI));
 75 | }
 76 |
 77 | double degrees_to_radians(double degrees)
 78 | {
 79 |       normalize_degrees(&degrees);
 80 |       return (degrees * (PI / 180.0));
 81 | }
 82 |
 83 | double degrees_to_grade(double degrees)
 84 | {
 85 |       normalize_degrees(&degrees);
 86 |       return (degrees * 1.11111111111111111111);
 87 | }
 88 |
 89 | double grade_to_radians(double grade)
 90 | {
 91 |       normalize_grade(&grade);
 92 |       return (grade * (PI / 200.0));
 93 | }
 94 |
 95 | double grade_to_degrees(double grade)
 96 | {
 97 |       normalize_grade(&grade);
 98 |       return (grade * 0.9);
 99 | }
100 |
```

```
101 | double sine(double angle, enum angle_type atype)
102 | {
103 |       switch (atype)
104 |       {
105 |       case RAD:
106 |             normalize_radians(&angle);
107 |             return sin(angle);
108 |             break;
109 |
110 |       case DEG:
111 |             return sin(degrees_to_radians(angle));
112 |             break;
113 |
114 |       case GRAD:
115 |             return sin(grade_to_radians(angle));
116 |       }
117 | }
118 |
119 | double cosine(double angle, enum angle_type atype)
120 | {
121 |       switch (atype)
122 |       {
123 |       case RAD:
124 |             normalize_radians(&angle);
125 |             return cos(angle);
126 |             break;
127 |
128 |       case DEG:
129 |             return cos(degrees_to_radians(angle));
130 |             break;
131 |
132 |       case GRAD:
133 |             return cos(grade_to_radians(angle));
134 |       }
135 | }
136 |
137 | double tangent(double angle, enum angle_type atype)
138 | {
139 |       switch (atype)
140 |       {
141 |       case RAD:
142 |             normalize_radians(&angle);
143 |             return tan(angle);
144 |             break;
145 |
146 |       case DEG:
147 |             return tan(degrees_to_radians(angle));
148 |             break;
149 |
150 |       case GRAD:
151 |             return tan(grade_to_radians(angle));
152 |       }
153 | }
154 |
155 | double secant(double angle, enum angle_type atype)
156 | {
157 |       return (1.0 / cosine(angle, atype));
158 | }
159 |
160 | double cosecant(double angle, enum angle_type atype)
161 | {
162 |       return (1.0 / sine(angle, atype));
163 | }
164 |
165 | double cotangent(double angle, enum angle_type atype)
166 | {
167 |       return (1.0 / tangent(angle, atype));
168 | }
169 |
170 | double arc_sine(double x, enum angle_type out_type)
171 | {
172 |       switch (out_type)
173 |       {
174 |       case RAD:
175 |             return asin(x);
176 |             break;
177 |
178 |       case DEG:
179 |             return radians_to_degrees(asin(x));
180 |             break;
181 |
182 |       case GRAD:
183 |             return radians_to_grade(asin(x));
184 |       }
185 | }
186 |
187 | double arc_cosine(double x, enum angle_type out_type)
188 | {
189 |       switch (out_type)
190 |       {
191 |       case RAD:
192 |             return acos(x);
193 |             break;
194 |
195 |       case DEG:
196 |             return radians_to_degrees(acos(x));
197 |             break;
198 |
199 |       case GRAD:
200 |             return radians_to_grade(acos(x));
```

```
201 |       }
202 | }
203 |
204 | double arc_tangent(double x, enum angle_type out_type)
205 | {
206 |       switch (out_type)
207 |       {
208 |       case RAD:
209 |             return atan(x);
210 |             break;
211 |
212 |       case DEG:
213 |             return radians_to_degrees(atan(x));
214 |             break;
215 |
216 |       case GRAD:
217 |             return radians_to_grade(atan(x));
218 |       }
219 | }
220 |
221 | double arc_secant(double x, enum angle_type out_type)
222 | {
223 |       return arc_cosine((1.0 / x), out_type);
224 | }
225 |
226 | double arc_cosecant(double x, enum angle_type out_type)
227 | {
228 |       return arc_sine((1.0 / x), out_type);
229 | }
230 |
231 | double arc_cotangent(double x, enum angle_type out_type)
232 | {
233 |       return arc_tangent((1.0 / x), out_type);
234 | }
235 |
236 | double hyperbolic_sine(double angle, enum angle_type atype)
237 | {
238 |       switch (atype)
239 |       {
240 |       case RAD:
241 |             normalize_radians(&angle);
242 |             return sinh(angle);
243 |             break;
244 |
245 |       case DEG:
246 |             return sinh(degrees_to_radians(angle));
247 |             break;
248 |
249 |       case GRAD:
250 |             return sinh(grade_to_radians(angle));
251 |       }
252 | }
253 |
254 | double hyperbolic_cosine(double angle, enum angle_type atype)
255 | {
256 |       switch (atype)
257 |       {
258 |       case RAD:
259 |             normalize_radians(&angle);
260 |             return cosh(angle);
261 |             break;
262 |
263 |       case DEG:
264 |             return cosh(degrees_to_radians(angle));
265 |             break;
266 |
267 |       case GRAD:
268 |             return cosh(grade_to_radians(angle));
269 |       }
270 | }
271 |
272 | double hyperbolic_tangent(double angle, enum angle_type atype)
273 | {
274 |       switch (atype)
275 |       {
276 |       case RAD:
277 |             normalize_radians(&angle);
278 |             return tanh(angle);
279 |             break;
280 |
281 |       case DEG:
282 |             return tanh(degrees_to_radians(angle));
283 |             break;
284 |
285 |       case GRAD:
286 |             return tanh(grade_to_radians(angle));
287 |       }
288 | }
289 |
290 | double hyperbolic_secant(double angle, enum angle_type atype)
291 | {
292 |       return (1.0 / hyperbolic_cosine(angle, atype));
293 | }
294 |
295 | double hyperbolic_cosecant(double angle, enum angle_type atype)
296 | {
297 |       return (1.0 / hyperbolic_sine(angle, atype));
298 | }
299 |
300 | double hyperbolic_cotangent(double angle, enum angle_type atype)
```

```
{
	return (1.0 / hyperbolic_tangent(angle, atype));
}

double hyperbolic_arc_sine(double x, enum angle_type out_type)
{
	switch (out_type)
	{
	case RAD:
		return asinh(x);
		break;

	case DEG:
		return radians_to_degrees(asinh(x));
		break;

	case GRAD:
		return radians_to_grade(asinh(x));
	}
}

double hyperbolic_arc_cosine(double x, enum angle_type out_type)
{
	switch (out_type)
	{
	case RAD:
		return acosh(x);
		break;

	case DEG:
		return radians_to_degrees(acosh(x));
		break;

	case GRAD:
		return radians_to_grade(acosh(x));
	}
}

double hyperbolic_arc_tangent(double x, enum angle_type out_type)
{
	switch (out_type)
	{
	case RAD:
		return atanh(x);
		break;

	case DEG:
		return radians_to_degrees(atanh(x));
		break;

	case GRAD:
		return radians_to_grade(atanh(x));
	}
}

double hyperbolic_arc_secant(double x, enum angle_type out_type)
{
	return hyperbolic_arc_cosine((1.0 / x), out_type);
}

double hyperbolic_arc_cosecant(double x, enum angle_type out_type)
{
	return hyperbolic_arc_sine((1.0 / x), out_type);
}

double hyperbolic_arc_cotangent(double x, enum angle_type out_type)
{
	return hyperbolic_arc_tangent((1.0 / x), out_type);
}

double sech(double x)
{
	return (2.0 / (exp(x) + exp(-x)));
}

double csch(double x)
{
	return (2.0 / (exp(x) - exp(-x)));
}

double coth(double x)
{
	return (exp(-x) / (exp(x) - exp(-x)) * 2.0 + 1.0);
}

double acoth(double x)
{
	return (log((x + 1.0) / (x - 1.0)) / 2.0);
}
```

TEXT STATISTICS

8605 characters
389 lines

LEXICAL STATISTICS

17 comments [std-C]
0 comments [C++]
6 preprocessor instructions
0 constants [character]
1 constants [string]
41 constants [numeric]

SYMBOL TABLE

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************/
/* Filespec  :  triglib.c triglib.h                  */
/* Date      :  February 21 1997                     */
/* Time      :  14:11                                */
/* Revision  :  1.0C                                 */
/* Update    :                                       */
/*****************************************************/
/* Programmer:  Nigel Traves                         */
/* Address   :  5 Breamer Road, Collingham, Newark,  */
/*           :  Notts, U.K.                          */
/* Post Code :  NG23 7PN                             */
/*****************************************************/
/* Released to the Public Domain                     */
/*****************************************************/

#ifndef TRIGLIB_H
#define TRIGLIB_H

/*********************************************************************/
/* This library is concerned entirely with angles in general and     */
/* trigonometric functions in particular.                            */
/*********************************************************************/

enum angle_type {RAD, DEG, GRAD};

/*********************************************************************/
/* The following three routines 'normalize' the supplied angle to    */
/* be within limits appropiate for the trigonemetric routines.       */
/* normalize_radians ensures that the supplied angle is between -PI  */
/* and +PI, normalize_degrees between -180.0 and +180.0 and          */
/* normalize_grade between -200.0 and +200.0.  NOTE - ALL the        */
/* normal trigonometric functions normalize the angle before use,    */
/* and the inverse functions after.                                  */
/*********************************************************************/

void normalize_radians(double *radians);
void normalize_degrees(double *degrees);
void normalize_grade(double *grade);

/*********************************************************************/
/* These six routines enable conversion, of angles, between          */
/* radians, degrees and grade.  NOTE there is no need to normalize   */
/* the angle to be converted before calling any of these routines    */
/* as they all call the appropriate normalisation routine.           */
/*********************************************************************/

double radians_to_degrees(double radians);
double radians_to_grade(double radians);
double degrees_to_radians(double degrees);
double degrees_to_grade(double degrees);
double grade_to_radians(double grade);
double grade_to_degrees(double grade);

/*********************************************************************/
/* The following six routines are the normal trigonometric           */
/* functions.                                                        */
/*********************************************************************/

double sine(double angle, enum angle_type atype);
double cosine(double angle, enum angle_type atype);
double tangent(double angle, enum angle_type atype);
double secant(double angle, enum angle_type atype);
double cosecant(double angle, enum angle_type atype);
double cotangent(double angle, enum angle_type atype);

/*********************************************************************/
/* The following six routines are the normal inverse trigonometric   */
/* functions.                                                        */
/*********************************************************************/

double arc_sine(double x, enum angle_type outtype);
double arc_cosine(double x, enum angle_type outtype);
double arc_tangent(double x, enum angle_type outtype);
double arc_secant(double x, enum angle_type outtype);
double arc_cosecant(double x, enum angle_type outtype);
double arc_cotangent(double x, enum angle_type outtype);

/*********************************************************************/
/* The following six routines are the hyperbolic trigonometric       */
/* functions.                                                        */
/*********************************************************************/

double hyperbolic_sine(double angle, enum angle_type atype);
double hyperbolic_cosine(double angle, enum angle_type atype);
double hyperbolic_tangent(double angle, enum angle_type atype);
double hyperbolic_secant(double angle, enum angle_type atype);
double hyperbolic_cosecant(double angle, enum angle_type atype);
double hyperbolic_cotangent(double angle, enum angle_type atype);

/*********************************************************************/
/* The following six routines are the hyperbolic inverse             */
/* trigonometric functions.                                          */
/*********************************************************************/

double hyperbolic_arc_sine(double x, enum angle_type outtype);
double hyperbolic_arc_cosine(double x, enum angle_type outtype);
double hyperbolic_arc_tangent(double x, enum angle_type outtype);
double hyperbolic_arc_secant(double x, enum angle_type outtype);
```

```
 101 | double hyperbolic_arc_cosecant(double x, enum angle_type outtype);
 102 | double hyperbolic_arc_cotangent(double x, enum angle_type outtype);
 103 |
 104 | /*********************************************************************/
 105 | /* The following four routines "complete" the standard library      */
 106 | /* functions.                                                        */
 107 | /*********************************************************************/
 108 |
 109 | double sech(double x);
 110 | double csch(double x);
 111 | double coth(double x);
 112 | double acoth(double x);
 113 |
 114 | #endif
```

TEXT STATISTICS

```
  5696 characters
   114 lines
```

LEXICAL STATISTICS

```
    54 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DEG             26
    GRAD            26
    RAD             26
    TRIGLIB_H       18      19
    acoth          112
    angle           61      62      63      64      65      66      85      86      87      88      89
           90
    angle_type              26      61      62      63      64      65      66      73      74      75
           76      77      78      85      86      87      88      89      90      97      98      99
          100     101     102
    arc_cosecant            77
    arc_cosine              74
    arc_cotangent           78
    arc_secant              76
    arc_sine        73
    arc_tangent             75
    atype           61      62      63      64      65      66      85      86      87      88      89
           90
    cosecant        65
    cosine          62
    cotangent       66
    coth           111
    csch           110
    degrees         39      51      52
    degrees_to_grade        52
    degrees_to_radians              51
    grade           40      53      54
    grade_to_degrees        54
    grade_to_radians        53
    hyperbolic_arc_cosecant         101
    hyperbolic_arc_cosine            98
    hyperbolic_arc_cotangent        102
    hyperbolic_arc_secant           100
    hyperbolic_arc_sine              97
    hyperbolic_arc_tangent           99
    hyperbolic_cosecant              89
    hyperbolic_cosine       86
    hyperbolic_cotangent             90
    hyperbolic_secant       88
    hyperbolic_sine         85
    hyperbolic_tangent               87
    normalize_degrees       39
    normalize_grade         40
    normalize_radians       38
    outtype         73      74      75      76      77      78      97      98      99     100     101
          102
    radians         38      49      50
    radians_to_degrees               49
    radians_to_grade        50
    secant          64
    sech           109
    sine            61
    tangent         63
    x       73      74      75      76      77      78      97      98      99     100     101     102
           109     110     111     112
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  TRIM.C - Remove leading, trailing, & excess embedded spaces
**
**  public domain by Bob Stout & Michael Dehlwes
*/

#include <ctype.h>
#include "snip_str.h"

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

char *trim (char *str)
{
      char *ibuf, *obuf;

      if (str)
      {
            for (ibuf = obuf = str; *ibuf; )
            {
                  while (*ibuf && (isspace (*ibuf)))
                        ibuf++;
                  if (*ibuf && (obuf != str))
                        *(obuf++) = ' ';
                  while (*ibuf && (!isspace (*ibuf)))
                        *(obuf++) = *(ibuf++);
            }
            *obuf = NUL;
      }
      return (str);
}

#if defined(__cplusplus) && __cplusplus
 }
#endif

#ifdef TEST

#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
      if (argc == 2)
      {
            printf ("trim(\"%s\")\n", argv[1]);
            printf ("returned \"%s\"\n", trim (argv[1]));
      }
      else
      {
            fprintf (stderr, "To test this function, call TRIM\n");
            fprintf (stderr, "with an argument enclosed in quotes.\n");
            fprintf (stderr, "   Example:\n");
            fprintf (stderr, "   C:\\>trim \"  test   test   \"\n");
            fprintf (stderr, "   trim(\"  test   test   \"\n");
            fprintf (stderr, "   returned \"test test\"\n\n");
            fprintf (stderr, "   C:\\>_\n");
            return EXIT_FAILURE;
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

TEXT STATISTICS

```
  1630 characters
    66 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     1 constants [character]
    11 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    EXIT_FAILURE            61
    EXIT_SUCCESS            63
    NUL             31
    TEST            40
    __cplusplus             12+     36+
    argc            45      47
    argv            45      49      50
    ctype            9
    defined         12      36
    fprintf         54      55      56      57      58      59      60
    h        9      42      43
    ibuf            18      22+     24+     25      26      28+     29
    isspace         24      28
    main            45
    obuf            18      22      26      27      29      31
    printf          49      50
    stderr          54      55      56      57      58      59      60
    stdio           42
    stdlib          43
    str             16      20      22      26      33
    trim            16      50
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** tname.c -- wrapper for the undocumented DOS function TRUENAME
**
** TRUENAME: interrupt 0x21 function 0x60
**
**   Call with: ah    =  60h
**              es:di -> destination buffer
**              ds:si -> source buffer
**
**   Returns:   carry bit set if there were problems
**
** This code hereby contributed to the public domain.
*/

#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>
#include "dosfiles.h"
#include "snip_str.h"         /* For trim()     */

/*
** Truename itself. Note that I'm using intdosx() rather than
** playing with some inline assembler -- I've discovered some
** people that actually don't have an assembler, poor bastards :-)
*/

char *Truename(char *dst, char *src)
{
      union REGS rg;
      struct SREGS rs;

      if (!src || !*src || !dst)
            return NULL;

      src = trim(src);

      rg.h.ah = 0x60;
      rg.x.si = FP_OFF(src);
      rg.x.di = FP_OFF(dst);
      rs.ds   = FP_SEG(src);
      rs.es   = FP_SEG(dst);

      intdosx(&rg, &rg, &rs);

      return (rg.x.cflag) ? NULL : dst;
}

#ifdef TEST

/*
** ... and a little test function.
*/

int main(int argc, char *argv[])
{
      char buf[128]="                              ", *s;
      int i;

      if (3 > _osmajor)
      {
            puts("Only works with DOS 3+");
            return -1;
      }
      if(argc > 1)
      {
            for(i = 1; i < argc; i++)
            {
                  s = Truename((char *)buf,(char *)argv[i]);
                  printf("%s=%s\n",argv[i], s ? buf : "(null)");
            }
      }
      else  printf("Usage: TRUENAME [filename [filename...]]\n");

      return 0;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1794 characters
    80 lines

LEXICAL STATISTICS

     6 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     7 constants [string]
     7 constants [numeric]

SYMBOL TABLE

    FP_OFF          41      42
    FP_SEG          43      44
    NULL            36      48
    REGS            32
    SREGS           33
    TEST            51
    Truename        30      71
    _osmajor        62
    ah              40
    argc            57      67      69
    argv            57      71      72
    buf             59      71      72
    cflag           48
    ctype           19
    di              42
    dos             20
    ds              43
    dst             30      35      42      44      48
    es              44
    h       17      18      19      20      40
    i       60      69+     71      72
    intdosx         46
    main            57
    printf          72      75
    puts            64
    rg              32      40      41      42      46+     48
    rs              33      43      44      46
    s       59      71      72
    si              41
    src             30      35+     38+     41      43
    stdlib          17
    string          18
    trim            38
    x       41      42      48
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Remove all files older than a given date from the current directory
**  and (optionally) all subdirectories
**
**  public domain by Bob Stout
**
**  uses SCALDATE.C & FTIME.C from SNIPPETS
**  also uses TODAY.C & FDATE.C from SNIPPETS
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <dos.h>
#include <ctype.h>
#include <time.h>
#include "sniptype.h"
#include "dirport.h"
#include "ftime.h"
#include "scaldate.h"
#if defined(MSDOS) || defined(__MSDOS__)
 #include "unistd.h"
#else
 #include <unistd.h>
#endif

#if (defined(_MSC_VER) && (_MSC_VER >= 700)) || (defined(__SC__))
 /* Make FP_xxx macros lvalues as in older versions */
 #undef FP_SEG
 #undef FP_OFF
 #define FP_SEG(fp)    ((unsigned)((unsigned long)(fp) >> 16))
 #define FP_OFF(fp)    ((unsigned)(fp && 0xffff))
#endif

#define show(s) fputs((s), stderr)

/*
**  Switches and globals
*/

Boolean_T recurse = False_, gobble = False_;
Boolean_T ignore = False_, noaction = False_;

long kill_date = 0L;
FILE *log = NULL;
long file_date;

/*
**  Local prototypes
*/

static void usage(Boolean_T errstat);
static void clean_dir(char *path);
static void do_dir(char *path);
static void RMfile(char *fname);
static void RMdir(char *dname);
static int  log_printf(char *str, char *arg);

/*
**  T_CLEAN - Deletes all files and (optionally) subdirectories
*/

int main(int argc, char *argv[])
{
      int i, j, span;
      Boolean_T found_dir = False_;
      void (*clean_func)(char *) = clean_dir;

      for (i = 1; i < argc; ++i)          /* Check for switches         */
      {
            if (NULL == strchr("-/", *argv[i]))
                  continue;               /* Assume it's a filename     */
            for (j = 1; argv[i][j] ; ++j) /* Traverse nested switches   */
            {
                  switch (toupper(argv[i][j]))
                  {
                  case 'R':
                        clean_func = do_dir;
                        break;

                  case 'G':
                        gobble = True_;
                        break;

                  case 'I':
                        ignore = True_;
                        break;

                  case 'N':
                        noaction = True_;
                        break;

                  case '?':
                        fputs("*** Help ***\n", stderr);
                        usage(False_);
                        break;

```

```
101 |                   case 'L':
102 |                         if (0 == strlen(&argv[i][++j]))
103 |                               log = stdout;
104 |                         else
105 |                         {
106 |                               log = fopen(strupr(&argv[i][j]), "w");
107 |                               if (NULL == log)
108 |                               {
109 |                                     fprintf(stderr, "*** Unable to open "
110 |                                           "logfile, %s\n", &argv[i][j]);
111 |                                     return Error_;
112 |                               }
113 |                         }
114 |                         j += strlen(&argv[i][j]) - 1; /* End of switch  */
115 |                         break;
116 |
117 |                   case 'T':
118 |                         if (0 == strlen(&argv[i][++j]))
119 |                         {
120 |                               fputs("*** No time specified ***\n", stderr);
121 |                               usage(Error_);               /* Oops     */
122 |                         }
123 |                         span = atoi(&argv[i][j]);
124 |                         kill_date = today() - (long)(span);
125 |                         j += strlen(&argv[i][j]) - 1; /* End of switch  */
126 |                         break;
127 |
128 |                   default:
129 |                         fprintf(stderr, "*** Bad switch - %c ***",
130 |                               argv[i][j]);
131 |                         usage(Error_);
132 |                   }
133 |             }
134 |       }
135 |       if (0L == kill_date)
136 |             kill_date = today() - 14L;
137 |
138 |       for (i = 1; i < argc; ++i)          /* Scan filenames             */
139 |       {
140 |             if (strchr("/-", *argv[i]))
141 |                   continue;
142 |             found_dir = True_;
143 |             clean_func(argv[i]);
144 |       }
145 |       if (!found_dir)
146 |       {
147 |             fputs("*** No directory specified ***\n", stderr);
148 |             usage(True_);
149 |       }
150 |       return EXIT_SUCCESS;
151 | }
152 |
153 | /*
154 | **  Tell 'em they messed up
155 | */
156 |
157 | static void usage(Boolean_T errstat)
158 | {
159 |       if (errstat)
160 |             fputc('\a', stderr);
161 |       show("Usage: T_CLEAN directory [...directory] [-tN] [-l[LOGFILE]] "
162 |             "[-rgin?]\n");
163 |       show("switches: -tN          Remove only files older than N days\n"
164 |             "                       (default = 14 days)\n");
165 |       show("          -l[LOGFILE]  Create named log file or (default) log to "
166 |             "the console\n");
167 |       show("          -r           Recurse subdirectories\n");
168 |       show("          -g           Gobble (delete) empty subdirectories\n");
169 |       show("          -i           Ignore special file attributes "
170 |             "(CAUTION!)\n");
171 |       show("          -n           No Action, display/log proposed "
172 |             "deletions\n");
173 |       show("          -?           Display help (this message)\n");
174 |       exit(errstat);
175 | }
176 |
177 | /*
178 | **  Clean all files from a directory
179 | */
180 |
181 | static void clean_dir(char *path)
182 | {
183 |       char rmpath[FILENAME_MAX], *rmfile;
184 |       DOSFileData fbuf;
185 |       unsigned attrib = (ignore) ? _A_ANY : 0;
186 |
187 |       strcpy(rmpath, path);
188 |       if ('\\' != LAST_CHAR(rmpath))
189 |             strcat(rmpath, "\\");
190 |       rmfile = &rmpath[strlen(rmpath)];
191 |       strcpy(rmfile, "*.*");
192 |       if (0 == FIND_FIRST(rmpath, attrib, &fbuf)) do
193 |       {
194 |             strcpy(rmfile, ff_name(&fbuf));
195 |             if (0 != getdatef(rmpath, &file_date))
196 |             {
197 |                   log_printf("Unable to get date of %s\n", rmpath);
198 |                   continue;
199 |             }
200 |             if (ignore)
```

```
201 |             {
202 |                   union REGS regs;
203 |                   struct SREGS sregs;
204 |
205 |                   regs.x.ax = 0x4300;
206 |                   regs.x.dx = FP_OFF((char FAR *)rmpath);
207 |                   segread(&sregs);
208 |                   sregs.ds  = FP_SEG((char FAR *)rmpath);
209 |                   intdosx(&regs, &regs, &sregs);
210 |                   if (!regs.x.cflag)
211 |                   {
212 |                         regs.x.ax  = 0x4301;
213 |                         regs.x.cx &= ~(_A_RDONLY | _A_HIDDEN | _A_SYSTEM);
214 |                         intdosx(&regs, &regs, &sregs);
215 |                         if (regs.x.cflag)
216 |                               log_printf("Unable to ignore attributes "
217 |                                     "of %s\n", rmpath);
218 |                   }
219 |             }
220 |             if (kill_date > file_date)
221 |                   RMfile(rmpath);
222 |       } while (0 == FIND_NEXT(&fbuf));
223 | }
224 |
225 | /*
226 | **  Process directories
227 | */
228 |
229 | static void do_dir(char *path)
230 | {
231 |       char search[FILENAME_MAX], new[FILENAME_MAX];
232 |       DOSFileData ff;
233 |
234 |       strcpy(search, path);
235 |       if ('\\' != LAST_CHAR(search))
236 |             strcat(search, "\\");
237 |       strcat(search, "*.*");
238 |       if (Success_ == FIND_FIRST(search, _A_ANY, &ff)) do
239 |       {
240 |             if (ff.attrib & _A_SUBDIR && '.' != *ff_name(&ff))
241 |             {
242 |                   strcpy(new, path);
243 |                   if ('\\' != LAST_CHAR(new))
244 |                         strcat(new, "\\");
245 |                   strcat(new, ff_name(&ff));
246 |                   do_dir(new);
247 |             }
248 |       } while (Success_ == FIND_NEXT(&ff));
249 |       clean_dir(path);
250 |       if (gobble)
251 |             RMdir(path);
252 | }
253 |
254 | /*
255 | **  File removal function - includes noaction switch and logging
256 | */
257 |
258 | static void RMfile(char *fname)
259 | {
260 |       unsigned yr, mo, dy;
261 |
262 |       if (!noaction)
263 |             remove(fname);
264 |       if (log)
265 |       {
266 |             scalar_to_ymd(file_date, &yr, &mo, &dy);
267 |             fprintf(log, "%02d/%02d/%04d: %s\n", mo, dy, yr, fname);
268 |       }
269 |       else  fprintf(stderr, "Deleting %s\n", fname);
270 | }
271 |
272 | /*
273 | **  Directory removal function - includes noaction switch and logging
274 | */
275 |
276 | static void RMdir(char *dname)
277 | {
278 |       if (!noaction)
279 |             rmdir(dname);
280 |       if (log)
281 |             fprintf(log, "Attempting to remove directory %s\n", dname);
282 | }
283 |
284 | /*
285 | **  Error logging printf()
286 | */
287 |
288 | static int log_printf(char *str, char *arg)
289 | {
290 |       FILE *out;
291 |
292 |       if (log)
293 |             out = log;
294 |       else  out = stderr;
295 |
296 |       return fprintf(out, str, arg);
297 | }
```

TEXT STATISTICS

8646 characters
297 lines

LEXICAL STATISTICS

19 comments [std-C]
0 comments [C++]
23 preprocessor instructions
12 constants [character]
38 constants [string]
19 constants [numeric]

SYMBOL TABLE

```
Boolean_T        44      45      55      69     157
DOSFileData             184     232
EXIT_SUCCESS            150
Error_          111     121     131
FAR             206     208
FILE             48     290
FILENAME_MAX            183     231+
FIND_FIRST              192     238
FIND_NEXT       222     248
FP_OFF           33      35     206
FP_SEG           32      34     208
False_           44+     45+     69      98
LAST_CHAR       188     235     243
MSDOS            24
NULL             48      74     107
REGS            202
RMdir            59     251     276
RMfile           58     221     258
SREGS           203
Success_        238     248
True_            85      89      93     142     148
_A_ANY          185     238
_A_HIDDEN       213
_A_RDONLY       213
_A_SUBDIR       240
_A_SYSTEM       213
_MSC_VER         30+
__MSDOS__        24
__SC__           30
arg              60     288     296
argc             66      72     138
argv             66      74      76      78     102     106     110     114     118     123     125
        130     140     143
atoi            123
attrib          185     192     240
ax              205     212
cflag           210     215
clean_dir        56      70     181     249
clean_func               70      81     143
ctype            18
cx              213
defined          24+     30+
dname            59     276     279     281
do_dir           57      81     229     246
dos              17
ds              208
dx              206
dy              260     266     267
errstat          55     157     159     174
exit            174
fbuf            184     192     194     222
ff              232     238     240+    245     248
ff_name         194     240     245
file_date        49     195     220     266
fname            58     258     263     267     269
fopen           106
found_dir        69     142     145
fp               34+     35+
fprintf         109     129     267     269     281     296
fputc           160
fputs            38      97     120     147
getdatef        195
gobble           44      85     250
h       13       14      15      16      17      18      19      27
i       68       72+     74      76      78     102     106     110     114     118     123     125
        130     138+    140     143
ignore           45      89     185     200
intdosx         209     214
io               16
j       68       76+     78     102     106     110     114+    118     123     125+    130
kill_date        47     124     135     136     220
log              48     103     106     107     264     267     280     281     292     293
log_printf               60     197     216     288
main             66
mo              260     266     267
noaction         45      93     262     278
out             290     293     294     296
path             56      57     181     187     229     234     242     249     251
recurse          44
regs            202     205     206     209+    210     212     213     214+    215
remove          263
rmdir           279
rmfile          183     190     191     194
rmpath          183     187     188     189     190+    192     195     197     206     208     217
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  UNISTD.H - Posix-compliant header for porting code to DOS
  5 | **             systems which lack this standard header file.
  6 | **
  7 | **  public domain by Bob Stout
  8 | */
  9 |
 10 | #ifndef __UNISTD_H      /* Use same macro as Symantec to avoid collision */
 11 | #define __UNISTD_H
 12 |
 13 |
 14 | #include <io.h>
 15 | #include <process.h>
 16 | #include <stdio.h>
 17 | #include "stat.h"
 18 |
 19 | #ifndef F_OK
 20 |  #define F_OK    0      /* File exists          */
 21 |  #define X_OK    1      /* File is executable   */
 22 |  #define W_OK    2      /* File is writable     */
 23 |  #define R_OK    4      /* File is readable     */
 24 | #endif
 25 |
 26 | #undef dirent
 27 | #undef DIR
 28 |
 29 | #if !defined(__TURBOC__)
 30 |  #include <direct.h>
 31 | #else
 32 |  #include <dir.h>
 33 | #endif
 34 |
 35 | #define dirent DIRENT_
 36 | #define DIR DIR_
 37 |
 38 |
 39 | #endif /* __UNISTD_H */
```

TEXT STATISTICS

```
   812 characters
    39 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
    22 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]
```

SYMBOL TABLE

```
    DIR             27      36
    DIRENT_         35
    DIR_            36
    F_OK            19      20
    R_OK            23
    W_OK            22
    X_OK            21
    __TURBOC__              29
    __UNISTD_H              10      11
    defined         29
    dir             32
    direct          30
    dirent          26      35
    h       14      15      16      30      32
    io              14
    process         15
    stdio           16
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  UNIX2DOS.C - Convert Unix-style pathnames to DOS-style
  5 | **
  6 | **  public domain by Bob Stout
  7 | */
  8 |
  9 | #include "filnames.h"
 10 |
 11 | char *unix2dos(char *path)
 12 | {
 13 |       char *p;
 14 |
 15 |       for (p = path; *p; ++p)
 16 |             if ('/' == *p)
 17 |                   *p = '\\';
 18 |       return path;
 19 | }
 20 |
 21 | char *dos2unix(char *path)
 22 | {
 23 |       char *p;
 24 |
 25 |       for (p = path; *p; ++p)
 26 |             if ('\\' == *p)
 27 |                   *p = '/';
 28 |       return path;
 29 | }
```

TEXT STATISTICS

```
   499 characters
    29 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     4 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    dos2unix        21
    p       13      15+     16      17      23      25+     26      27
    path            11      15      18      21      25      28
    unix2dos        11
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  UNXCONIO.H - Port crucial DOS|Win|OS/2 non-blocking console I/O
  5 | **               functions to Unix/Posix.
  6 | **
  7 | **  public domain SNIPPETS header for use with Steve Poole's TERM_OPT.C
  8 | */
  9 |
 10 | #ifndef UNXCONIO__H
 11 | #define UNXCONIO__H
 12 |
 13 | #include <stdio.h>
 14 | #include <unistd.h>
 15 |
 16 | #define echo_on()  term_option(0)
 17 | #define echo_off() term_option(1)
 18 |
 19 | int term_option();
 20 | int getch();
 21 |
 22 | #endif /* UNXCONIO__H */
```

TEXT STATISTICS

```
   464 characters
    22 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     0 constants [string]
     2 constants [numeric]
```

SYMBOL TABLE

```
    UNXCONIO__H             10      11
    echo_off        17
    echo_on         16
    getch           20
    h       13      14
    stdio           13
    term_option             16      17      19
    unistd          14
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  UNXGETCH.C - Non-blocking console input function for Unix/Posix.
  5 | **
  6 | **  public domain by Bob Stout using Steve Poole's term_option().
  7 | */
  8 |
  9 | #include "unxconio.h"
 10 |
 11 | int getch()
 12 | {
 13 |       int istat, key;
 14 |       char buf[2];
 15 |
 16 |       if (0 > term_option(1))
 17 |             return EOF;
 18 |       if (0 > term_option(2))
 19 |             return EOF;
 20 |       istat = read(STDIN_FILENO,&buf,1);
 21 |       if (istat < 0)
 22 |             key = EOF;
 23 |       else  key = (int)buf[0];
 24 |       term_option(0);
 25 |       return key;
 26 | }
 27 |
 28 | #ifdef TEST
 29 |
 30 | #include <ctype.h>
 31 |
 32 | main()
 33 | {
 34 |       int key;
 35 |
 36 |       printf("Press any key to continue...");
 37 |       fflush(stdout);
 38 |       key = getch();
 39 |       printf("\n\nYou pressed \"%c\"\n", toupper(key));
 40 |       return 0;
 41 | }
 42 |
 43 | #endif /* TEST */
```

TEXT STATISTICS

```
   816 characters
    43 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     3 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    EOF             17      19      22
    STDIN_FILENO            20
    TEST            28
    buf             14      20      23
    ctype           30
    fflush          37
    getch           11      38
    h       30
    istat           13      20      21
    key             13      22      23      25      34      38      39
    main            32
    printf          36      39
    read            20
    stdout          37
    term_option             16      18      24
    toupper         39
```

```
/* +++Date last modified: 05-Jul-1997 */

/* microsleep.c */

/* Author:
 * Pieter de Jong
 * Released in Public Domain 7-1996
 */


/* This module contains the unix-specific versions the microsleep function.
 * There are actually two versions of microsleep defined here, because
 * BSD, SysV, and V7 all need quite different implementations.
 *
 * At this time, the select() call is not (yet) in the POSIX standard, but
 * it is in BSD4.2+ , and SVR4.
 * poll() is SYSV only. BSD has a usleep() function that looks & feels
 * like microsleep(), but implements it using setitimer(). It has
 * (usually) unwanted side effects when multiple timers are active.
 *
 * This version should interact correctly with other timers set
 * by the calling process, and is not interrupted if a signal is caught
 */
/* #include "config.h"*/

# ifdef BSD
/* For BSD, we use select() */

#include<sys/types.h>
#include<sys/time.h>
#include<stddef.h>

void
microsleep(unsigned int microsecs)
{
    struct timeval tval;

    tval.tv_sec = microsecs/ 1000000;
    tval.tv_usec= microsecs% 1000000;
    select(0, NULL, NULL, NULL, &tval);
}
#endif

# ifdef SYSV
/* For System-V, we use poll to implement the timeout. */
/* R4 has select(), but implements it using poll() */
/* older versions may only have poll() */

#include<sys/types.h>
#include<sys/poll.h>
#include<stropts.h>

void
microsleep(unsigned int microsecs)
{
    struct pollfd dummy;
    int timeout;

    if ((timeout = microsecs / 1000) <= 0)
        timeout = 1;
    poll(&dummy, 0, timeout);
}

# endif /* !BSD */
```

```
TEXT STATISTICS

  1595 characters
    64 lines

LEXICAL STATISTICS

    10 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
     0 constants [string]
     7 constants [numeric]

SYMBOL TABLE

    BSD             26
    NULL            40+
    SYSV            44
    dummy           56      61
    endif           64
    h       29      30      31      49      50      51
    ifdef           26      44
    microsecs       34      38      39      54      59
    microsleep              34      54
    poll            50      61
    pollfd          56
    select          40
    stddef          31
    stropts         51
    sys             29      30      49      50
    time            30
    timeout         57      59      60      61
    timeval         36
    tv_sec          38
    tv_usec         39
    tval            36      38      39      40
    types           29      49
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** update_file() - 1) Renames szFname.bak (if existed) to tmpnam
**                 2) Renames szFname to szFname.bak
**                 3) Renames szOther to szFname.
**                    If szOther == NULL, skip this step (REN2BAK only).
**                 4) Removes old bak (tmpnam) file.
**
** public domain by Phi Nguyen
*/

#include "update.h"

Boolean_T update_file(char *szFname, char *szOther)
{
      char  szBakname[FILENAME_MAX],
            szTbakname[L_tmpnam],
            *p;

      if (NULL == szFname)
            return Error_;

      strcpy(szBakname, szFname);
      p = strrchr(szBakname, '.');
      if (p && !strpbrk(p, "\\/"))
            *p = NUL;
      strcat(szBakname, ".bak");

      rename(szBakname, tmpnam(szTbakname));
      if (rename(szFname, szBakname))
      {
            rename(szTbakname, szBakname);
            return Error_;
      }

      if (NULL != szOther)
      {
            if (rename(szOther, szFname))
            {
                  rename(szBakname, szFname);
                  rename(szTbakname, szBakname);
                  return Error_;
            }
      }

      remove(szTbakname);
      return Success_;
}

#ifdef TEST

#include <stdlib.h>

main(int argc, char *argv[])
{
      int i;

      if (1 == argc)
      {
            puts("Usage: Ren2Bak filename ...");
            return EXIT_FAILURE;
      }

      for (i = 1; i < argc; i++)
      {
            if (Error_ == ren2bak(argv[i]))
            {
                  printf("Can't rename file '%s'.\n", argv[i]);
                  return EXIT_FAILURE;
            }
      }
      return EXIT_SUCCESS;
}

#endif /* TEST */
```

```
TEXT STATISTICS

  1766 characters
    76 lines

LEXICAL STATISTICS

     3 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     1 constants [character]
     5 constants [string]
     2 constants [numeric]

SYMBOL TABLE

    Boolean_T       15
    EXIT_FAILURE            62      70
    EXIT_SUCCESS            73
    Error_          22      34      43      67
    FILENAME_MAX            17
    L_tmpnam        18
    NUL             27
    NULL            21      37
    Success_        48
    TEST            51
    argc            55      59      65
    argv            55      67      69
    h       53
    i       57      65+     67      69
    main            55
    p       19      25      26+     27
    printf          69
    puts            61
    remove          47
    ren2bak         67
    rename          30      31      33      39      41      42
    stdlib          53
    strcat          28
    strcpy          24
    strpbrk         26
    strrchr         25
    szBakname       17      24      25      28      30      31      33      41      42
    szFname         15      21      24      31      39      41
    szOther         15      37      39
    szTbakname              18      30      33      42      47
    tmpnam          30
    update_file             15
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | ** UPDATE.H - Header for UPDATE.C
   5 | */
   6 |
   7 | #ifndef UPDATE__H
   8 | #define UPDATE__H
   9 |
  10 | #include <io.h>
  11 | #include <stdio.h>
  12 | #include <string.h>
  13 | #include <stdlib.h>
  14 | #include "sniptype.h"
  15 |
  16 | Boolean_T update_file(char *szFname, char *szOther);
  17 |
  18 | #define ren2bak(x) update_file(x, NULL)
  19 |
  20 | #endif /* UPDATE__H */
```

TEXT STATISTICS

```
   356 characters
    20 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     0 constants [character]
     1 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       16
    NULL            18
    UPDATE__H        7       8
    h       10      11      12      13
    io              10
    ren2bak         18
    stdio           11
    stdlib          13
    string          12
    szFname         16
    szOther         16
    update_file             16      18
    x       18+
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** by: John Lots
  5 | ** patched up for BC++ 3.1 by Alan Eldridge 10/12/92
  6 | **     (UUCP: alane@wozzle.linet.org, FIDO: 1:272/38.473)
  7 | */
  8 |
  9 | #include        <stdio.h>
 10 | #include        <stdlib.h>
 11 | #define         DEC(c)  (char)(((c)-' ')&077)
 12 |
 13 | int main()
 14 | {
 15 |       int     n;
 16 |       char    buf[128],a,b,c,d;
 17 |
 18 |       scanf("begin %o ", &n);
 19 |       gets(buf);                              /* filename */
 20 |       if (!freopen(buf, "wb", stdout))         /* oops.. */
 21 |       {
 22 |             perror(buf);
 23 |             exit(1);
 24 |       }
 25 |       while ((n=getchar())!=EOF&&((n=DEC(n))!=0))
 26 |       {
 27 |             while (n>0)
 28 |             {
 29 |                   a=DEC(getchar());
 30 |                   b=DEC(getchar());
 31 |                   c=DEC(getchar());
 32 |                   d=DEC(getchar());
 33 |                   if (n-->0)
 34 |                         putchar((a<<2)|(b>>4));
 35 |                   if (n-->0)
 36 |                         putchar((b<<4)|(c>>2));
 37 |                   if (n-->0)
 38 |                         putchar((c<<6)|d);
 39 |             }
 40 |             n=getchar();                    /* skip \n */
 41 |       }
 42 |       return 0;
 43 | }
```

TEXT STATISTICS

```
  1160 characters
    43 lines
```

LEXICAL STATISTICS

```
     5 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     1 constants [character]
     2 constants [string]
    14 constants [numeric]
```

SYMBOL TABLE

```
    DEC             11      25      29      30      31      32
    EOF             25
    a       16      29      34
    b       16      30      34      36
    buf             16      19      20      22
    c       11+     16      31      36      38
    d       16      32      38
    exit            23
    freopen         20
    getchar         25      29      30      31      32      40
    gets            19
    h        9      10
    main            13
    n       15      18      25+     27      33      35      37      40
    perror          22
    putchar         34      36      38
    scanf           18
    stdio            9
    stdlib          10
    stdout          20
```

```
/* +++Date last modified: 05-Jul-1997 */

/* uuencode.c */

/*
uudecode and uuencode are easily implemented under MSDOS as well.  Here
are the sources for Microsoft C v3.0, but if you have another kind of C
compiler, there should be perhaps only 1 change -- the output file of
uudecode and the input file of uuencode must be in binary format.
(ie.  binary files, like .EXE files may have byte patterns that are the
same as ^Z, which signals end-of-file in non-binary (text) mode).

    Don Kneller
UUCP:   ...ucbvax!ucsfcgl!kneller
ARPA:   kneller@ucsf-cgl.ARPA
BITNET: kneller@ucsfcgl.BITNET

   patched up for BC++ 3.1 by Alan Eldridge 10/12/92
       (UUCP: alane@wozzle.linet.org, FIDO: 1:272/38.473)

*/

#ifndef lint
 #if !defined(MSDOS) && !defined(__MSDOS__)
 static char sccsid[] = "@(#)uuencode.c  5.1 (Berkeley) 7/2/83";
 #endif
#endif

#if defined(MSDOS) || defined(__MSDOS__)
 #define READ_BINARY "rb"
#else
 #define READ_BINARY "r"
#endif

/*
 * uuencode [input] output
 *
 * Encode a file so it can be mailed to a remote system.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>

/* ENC is the basic 1 character encoding function to make a char printing */

#define ENC(c) (((c) & 077) + ' ')

void encode(FILE *in, FILE *out);
void outdec(char *p, FILE *f);
int fr(FILE *fd, char *buf, int cnt);

main(int argc, char *argv[])
{
      FILE *in;
      struct stat sbuf;
      int mode;

      /* optional 1st argument */

      if (argc > 2)
      {
            /* Use binary mode */
            if ((in = fopen(argv[1], READ_BINARY)) == NULL)
            {
                  perror(argv[1]);
                  exit(1);
            }
            argv++; argc--;
      }
      else  in = stdin;

      if (argc != 2)
      {
            printf("Usage: uuencode [infile] remotefile\n");
            exit(2);
      }

      /* figure out the input file mode */

      fstat(fileno(in), &sbuf);
      mode = sbuf.st_mode & 0777;
      printf("begin %o %s\n", mode, argv[1]);

      encode(in, stdout);

      printf("end\n");
      return 0;
}

/*
 * copy from in to out, encoding as you go along.
 */

void encode(FILE *in, FILE *out)
{
      char buf[80];
      int i, n;

```

```
101 |       for (;;)
102 |       {
103 |             /* 1 (up to) 45 character line */
104 |
105 |             n = fr(in, buf, 45);
106 |             putc(ENC(n), out);
107 |
108 |             for (i = 0; i < n; i += 3)
109 |                   outdec(&buf[i], out);
110 |
111 |             putc('\n', out);
112 |             if (n <= 0)
113 |                   break;
114 |       }
115 | }
116 |
117 | /*
118 |  * output one group of 3 bytes, pointed at by p, on file f.
119 |  */
120 |
121 | void outdec(char *p, FILE *f)
122 | {
123 |       int c1, c2, c3, c4;
124 |
125 |       c1 = *p >> 2;
126 |       c2 = ((p[0] << 4) & 060) | ((p[1] >> 4) & 017);
127 |       c3 = ((p[1] << 2) & 074) | ((p[2] >> 6) & 03);
128 |       c4 = p[2] & 077;
129 |       putc(ENC(c1), f);
130 |       putc(ENC(c2), f);
131 |       putc(ENC(c3), f);
132 |       putc(ENC(c4), f);
133 | }
134 |
135 | /* fr: like read but stdio */
136 |
137 | int fr(FILE *fd, char *buf, int cnt)
138 | {
139 |       int c, i;
140 |
141 |       for (i = 0; i < cnt; i++)
142 |       {
143 |             c = getc(fd);
144 |             if (c == EOF)
145 |                   return(i);
146 |             buf[i] = (char)c;
147 |       }
148 |       return (cnt);
149 | }
```

TEXT STATISTICS

```
  3321 characters
   149 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    14 preprocessor instructions
     2 constants [character]
     6 constants [string]
    31 constants [numeric]
```

SYMBOL TABLE

```
    ENC             48     106     129     130     131     132
    EOF            144
    FILE            50+     51      52      56      96+    121     137
    MSDOS           24      29
    NULL            65
    READ_BINARY             30      32      65
    __MSDOS__       24      29
    argc            54      62      70      74
    argv            54      65      67      70      84
    buf             52      98     105     109     137     146
    c       48+    139     143     144     146
    c1             123     125     129
    c2             123     126     130
    c3             123     127     131
    c4             123     128     132
    cnt             52     137     141     148
    defined         24+     29+
    encode          50      86      96
    exit            68      77
    f       51     121     129     130     131     132
    fd              52     137     143
    fileno          82
    fopen           65
    fr              52     105     137
    fstat           82
    getc           143
    h       41      42      43      44
    i       99     108+    109     139     141+    145     146
    in              50      56      65      72      82      86      96     105
    lint            23
    main            54
    mode            58      83      84
    n       99     105     106     108     112
    out             50      96     106     109     111
    outdec          51     109     121
    p       51     121     125     126+    127+    128
    perror          67
    printf          76      84      88
    putc           106     111     129     130     131     132
    sbuf            57      82      83
    sccsid          25
    st_mode         83
    stat            44      57
    stdin           72
    stdio           41
    stdlib          42
    stdout          86
    sys             43      44
    types           43
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  Functions to blank and resore EGA/VGA screens
  5 | */
  6 |
  7 | #include <dos.h>
  8 | #include "pchwio.h"
  9 | #include "vgablank.h"
 10 |
 11 | void blank_EVGA(void)
 12 | {
 13 |     disable();
 14 |     inp(0x3da);
 15 |     outp(0x3c0, 0);
 16 |     enable();
 17 | }
 18 |
 19 | void restore_EVGA(void)
 20 | {
 21 |     disable();
 22 |     inp(0x3da);
 23 |     outp(0x3c0, 0x20);
 24 |     enable();
 25 | }
 26 |
 27 | #ifdef TEST
 28 |
 29 | #include "delay.h"      /* From SNIPPETS - use dos.h if your compiler's
 30 |                            standard library supports delay().          */
 31 | main()
 32 | {
 33 |       blank_EVGA();     /* Blank the screen...  */
 34 |       delay(1500);      /* ...delay 1.5 secs... */
 35 |       restore_EVGA();   /* ...& restore it      */
 36 |       return 0;
 37 | }
 38 |
 39 | #endif /* TEST */
```

TEXT STATISTICS

```
   722 characters
    39 lines
```

LEXICAL STATISTICS

```
     7 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     3 constants [string]
     8 constants [numeric]
```

SYMBOL TABLE

```
    TEST            27
    blank_EVGA              11      33
    delay           34
    disable         13      21
    dos              7
    enable          16      24
    h       7
    inp             14      22
    main            31
    outp            15      23
    restore_EVGA            19      35
```

```
   1 | /* +++Date last modified: 05-Jul-1997 */
   2 |
   3 | /*
   4 | **  SNIPPETS header file for VGABLANK.C
   5 | */
   6 |
   7 | #ifndef VGABLANK__H
   8 | #define VGABLANK__H
   9 |
  10 | void blank_EVGA(void);
  11 | void restore_EVGA(void);
  12 |
  13 | #endif /* VGABLANK__H */
```

TEXT STATISTICS

```
   217 characters
    13 lines
```

LEXICAL STATISTICS

```
     3 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    VGABLANK__H              7       8
    blank_EVGA              10
    restore_EVGA            11
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VIDMGR.C; Screen drawing, cursor and keyboard routines for text mode
 *            16-bit and 32-bit MS-DOS, 16-bit and 32-bit OS/2, and 32-bit
 *            Windows 95/NT applications.  Release 1.3.
 *
 *  This module written in March 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in October 1996.
 *
 *  VidMgr has been compiled and tested with the following C compilers:
 *
 *    - Borland C++ (16-bit) for DOS 3.1
 *    - Borland C++ (16-bit) for DOS 4.5
 *    - Borland C++ (32-bit) for OS/2 1.0
 *    - Cygnus GNU C (32-bit) for Windows 95/NT b14.0
 *    - DJGPP GNU C (32-bit) for DOS 2.0
 *    - EMX GNU C (32-bit) for OS/2 & DOS 0.9b
 *    - IBM VisualAge C/C++ 3.0 (32-bit) for OS/2
 *    - Microsoft C/C++ (16-bit) for OS/2 6.00a
 *    - Microsoft C/C++ (16-bit) for DOS 8.00c
 *    - Microsoft Quick C (16-bit) for DOS 2.50
 *    - Microsoft Visual C/C++ (16-bit) for DOS 1.52
 *    - WATCOM C/C++ (16-bit & 32-bit) for DOS 9.5
 *    - WATCOM C/C++ (16-bit & 32-bit) for DOS 10.0
 *    - WATCOM C/C++ (32-bit) for OS/2 10.0
 *    - WATCOM C/C++ (32-bit) for Windows 95/NT 10.0
 *    - HI-TECH Pacific C (16-bit) for DOS 7.51
 *    - Symantec C/C++ (16-bit) for DOS 7.0
 *    - Zortech C/C++ (16-bit) for DOS 3.0r4
 */

#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include "vidmgr.h"

struct vm_info vm_startup;
char vm_curattr;

char vm_frame_blank[] = "      ";
char vm_frame_single[] = "ÚÄ¿³ÀÙ";
char vm_frame_double[] = "ÉÍ»ºÈ¼";

void vm_setattr(char attr)
{
    vm_curattr = attr;
}

void vm_printf(char x, char y, const char *format,...)
{
    va_list args;
    char buffer[512];
    va_start(args, format);
    vsprintf(buffer, format, args);
    va_end(args);
    vm_puts(x, y, buffer);
}

void vm_printfcenter(char row, const char *format,...)
{
    va_list args;
    char buffer[512];
    va_start(args, format);
    vsprintf(buffer, format, args);
    va_end(args);
    vm_puts((char)((vm_getscreenwidth() / 2) - (strlen(buffer) / 2)), row, buffer);
}

void vm_printfbetween(char x1, char x2, char row, const char *format,...)
{
    char x;
    va_list args;
    char buffer[512];
    va_start(args, format);
    vsprintf(buffer, format, args);
    va_end(args);
    if ((char)strlen(buffer) >= (char)(x2 - x1 + 1))
    {
        vm_puts(x1, row, buffer);
    }
    else
    {
        x = (char)(x1 + (x2 - x1 + 1 - strlen(buffer)) / 2);
        vm_puts(x, row, buffer);
    }
}

void vm_xprintf(char x, char y, char attr, const char *format,...)
{
    va_list args;
    char buffer[512];
    va_start(args, format);
    vsprintf(buffer, format, args);
    va_end(args);
    vm_xputs(x, y, attr, buffer);
}

void vm_xprintfcenter(char row, char attr, const char *format,...)
{
```

```
101 |     va_list args;
102 |     char buffer[512];
103 |     va_start(args, format);
104 |     vsprintf(buffer, format, args);
105 |     va_end(args);
106 |     vm_xputs((char)((vm_getscreenwidth() / 2) - (strlen(buffer) / 2)), row, attr, buffer);
107 | }
108 |
109 | void vm_xprintfbetween(char x1, char x2, char row, char attr, const char *format,...)
110 | {
111 |     char x;
112 |     va_list args;
113 |     char buffer[512];
114 |     va_start(args, format);
115 |     vsprintf(buffer, format, args);
116 |     va_end(args);
117 |     if ((char)strlen(buffer) >= (char)(x2 - x1 + 1))
118 |     {
119 |         vm_xputs(x1, row, attr, buffer);
120 |     }
121 |     else
122 |     {
123 |         x = (char)(x1 + (x2 - x1 + 1 - strlen(buffer)) / 2);
124 |         vm_xputs(x, row, attr, buffer);
125 |     }
126 | }
127 |
128 | void vm_paintclearscreen(char attr)
129 | {
130 |     vm_paintclearbox(1, 1, vm_getscreenwidth(), vm_getscreenheight(), attr);
131 | }
132 |
133 | void vm_paintclearline(char row, char attr)
134 | {
135 |     vm_paintclearbox(1, row, vm_getscreenwidth(), row, attr);
136 | }
137 |
138 | void vm_paintcleareol(char row, char attr)
139 | {
140 |     vm_paintclearbox(vm_wherex(), row, (char)(vm_getscreenwidth() - vm_wherex() + 1), row, attr);
141 | }
142 |
143 | void vm_paintscreen(char attr)
144 | {
145 |     vm_paintbox(1, 1, vm_getscreenwidth(), vm_getscreenheight(), attr);
146 | }
147 |
148 | void vm_paintline(char row, char attr)
149 | {
150 |     vm_paintbox(1, row, vm_getscreenwidth(), row, attr);
151 | }
152 |
153 | void vm_painteol(char row, char attr)
154 | {
155 |     vm_paintbox(vm_wherex(), row, (char)(vm_getscreenwidth() - vm_wherex() + 1), row, attr);
156 | }
157 |
158 | void vm_clearscreen(void)
159 | {
160 |     vm_clearbox(1, 1, vm_getscreenwidth(), vm_getscreenheight());
161 | }
162 |
163 | void vm_clearline(char row)
164 | {
165 |     vm_clearbox(1, row, vm_getscreenwidth(), row);
166 | }
167 |
168 | void vm_cleareol(char row)
169 | {
170 |     vm_clearbox(vm_wherex(), row, (char)(vm_getscreenwidth() - vm_wherex() + 1), row);
171 | }
172 |
173 | void vm_fillscreen(char ch)
174 | {
175 |     vm_fillbox(1, 1, vm_getscreenwidth(), vm_getscreenheight(), ch);
176 | }
177 |
178 | void vm_fillline(char row, char ch)
179 | {
180 |     vm_fillbox(1, row, vm_getscreenwidth(), row, ch);
181 | }
182 |
183 | void vm_filleol(char row, char ch)
184 | {
185 |     vm_fillbox(vm_wherex(), row, (char)(vm_getscreenwidth() - vm_wherex() + 1), row, ch);
186 | }
187 |
188 | void vm_clrscr(void)
189 | {
190 |     vm_paintclearscreen(vm_curattr);
191 |     vm_gotoxy(1, 1);
192 | }
193 |
194 | void vm_clreol(void)
195 | {
196 |     vm_paintcleareol(vm_wherey(), vm_curattr);
197 | }
198 |
199 | void vm_vertline(char y1, char y2, char col, char attr, char ch)
200 | {
```

```
 201 |     char y;
 202 |     for (y = y1; y <= y2; y++)
 203 |     {
 204 |         vm_xputch(col, y, attr, ch);
 205 |     }
 206 | }
 207 |
 208 | void vm_frame(char x1, char y1, char x2, char y2, char attr, char *frame)
 209 | {
 210 |     vm_xputch(x1, y1, attr, frame[0]);
 211 |     vm_horizline((char)(x1 + 1), (char)(x2 - 1), y1, attr, frame[1]);
 212 |     vm_xputch(x2, y1, attr, frame[2]);
 213 |     vm_vertline((char)(y1 + 1), (char)(y2 - 1), x1, attr, frame[3]);
 214 |     vm_vertline((char)(y1 + 1), (char)(y2 - 1), x2, attr, frame[3]);
 215 |     vm_xputch(x1, y2, attr, frame[4]);
 216 |     vm_horizline((char)(x1 + 1), (char)(x2 - 1), y2, attr, frame[1]);
 217 |     vm_xputch(x2, y2, attr, frame[5]);
 218 | }
```

TEXT STATISTICS

```
  5859 characters
   218 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     4 preprocessor instructions
     0 constants [character]
     4 constants [string]
    50 constants [numeric]
```

SYMBOL TABLE

```
    args            52      54      55      56      62      64      65      66      73      75      76
           77       91      93      94      95     101     103     104     105     112     114     115
          116
    attr            45      47      89      96      99     106     109     119     124     128     130
          133      135     138     140     143     145     148     150     153     155     199     204
          208      210     211     212     213     214     215     216     217
    buffer          53      55      57      63      65      67+     74      76      78      80      84
           85       92      94      96     102     104     106+    113     115     117     119     123
          124
    ch             173     175     178     180     183     185     199     204
    col            199     204
    format          50      54      55      60      64      65      70      75      76      89      93
           94       99     103     104     109     114     115
    frame          208     210     211     212     213     214     215     216     217
    h               33      34      35
    row             60      67      70      80      85      99     106     109     119     124     133
          135+     138     140+    148     150+    153     155+    163     165+    168     170+    178
          180+     183     185+
    stdarg          34
    stdio           33
    string          35
    strlen          67      78      84     106     117     123
    va_end          56      66      77      95     105     116
    va_list         52      62      73      91     101     112
    va_start        54      64      75      93     103     114
    vm_clearbox            160     165     170
    vm_cleareol            168
    vm_clearline           163
    vm_clearscreen         158
    vm_clreol      194
    vm_clrscr      188
    vm_curattr              39      47     190     196
    vm_fillbox             175     180     185
    vm_filleol             183
    vm_fillline            178
    vm_fillscreen          173
    vm_frame       208
    vm_frame_blank          41
    vm_frame_double         43
    vm_frame_single         42
    vm_getscreenheight             130     145     160     175
    vm_getscreenwidth       67     106     130     135     140     145     150     155     160     165
           170     175     180     185
    vm_gotoxy      191
    vm_horizline           211     216
    vm_info         38
    vm_paintbox            145     150     155
    vm_paintclearbox       130     135     140
    vm_paintcleareol       138     196
    vm_paintclearline      133
    vm_paintclearscreen            128     190
    vm_painteol            153
    vm_paintline           148
    vm_paintscreen         143
    vm_printf       50
    vm_printfbetween        70
    vm_printfcenter         60
    vm_puts         57      67      80      85
    vm_setattr              45
    vm_startup              38
    vm_vertline            199     213     214
    vm_wherex      140+    155+    170+    185+
    vm_wherey      196
    vm_xprintf              89
```

```
vm_xprintfbetween      109
vm_xprintfcenter        99
vm_xputch      204     210     212     215     217
vm_xputs        96     106     119     124
vsprintf        55      65      76      94     104     115
x       50      57      72      84      85      89      96     111     123     124
x1              70      78      80     84+     109     117     119    123+     208     210     211
        213     215     216
x2              70      78      84      109     117     123     208     211     212     214     216
        217
y       50      57      89      96      201    202+     204
y1             199     202     208     210     211     212     213     214
y2             199     202     208     213     214     215     216     217
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VIDMGR.H; Screen drawing, cursor and keyboard routines for text mode
 *            16-bit and 32-bit MS-DOS, 16-bit and 32-bit OS/2, and 32-bit
 *            Windows 95/NT applications.  Release 1.2.
 *
 *  This module written in March 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in June 1996.
 *
 *  VidMgr has been compiled and tested with the following C compilers:
 *
 *    - Borland C++ (16-bit) for DOS 3.1
 *    - Borland C++ (16-bit) for DOS 4.5
 *    - Borland C++ (32-bit) for OS/2 1.0
 *    - Cygnus GNU C (32-bit) for Windows 95/NT b14.0
 *    - DJGPP GNU C (32-bit) for DOS 2.0
 *    - EMX GNU C (32-bit) for OS/2 & DOS 0.9b
 *    - IBM VisualAge C/C++ 3.0 (32-bit) for OS/2
 *    - Microsoft C/C++ (16-bit) for OS/2 6.00a
 *    - Microsoft C/C++ (16-bit) for DOS 8.00c
 *    - Microsoft Quick C (16-bit) for DOS 2.50
 *    - Microsoft Visual C/C++ (16-bit) for DOS 1.52
 *    - WATCOM C/C++ (16-bit & 32-bit) for DOS 9.5
 *    - WATCOM C/C++ (16-bit & 32-bit) for DOS 10.0
 *    - WATCOM C/C++ (32-bit) for OS/2 10.0
 *    - WATCOM C/C++ (32-bit) for Windows 95/NT 10.0
 *    - HI-TECH Pacific C (16-bit) for DOS 7.51
 *    - Symantec C/C++ (16-bit) for DOS 7.0
 *    - Zortech C/C++ (16-bit) for DOS 3.0r4
 */

#ifndef __VIDMGR_H__
#define __VIDMGR_H__

#ifndef DOS
#if defined(_QC) || defined(__DOS__) || defined(MSDOS) || defined(__MSDOS__)
#define DOS
#endif
#endif

#ifndef OS2
#if defined(__OS2__) || defined(OS_2)
#define OS2
#endif
#endif

#ifndef EMX
#if defined(__EMX__)
#define EMX
#endif
#endif

#ifndef WINNT
#if defined(__NT__)
#define WINNT
#endif
#endif

#define BLACK         0x00
#define BLUE          0x01
#define GREEN         0x02
#define CYAN          0x03
#define RED           0x04
#define MAGENTA       0x05
#define PURPLE        MAGENTA
#define BROWN         0x06
#define LIGHTGRAY     0x07
#define LIGHTGREY     LIGHTGRAY
#define DARKGRAY      0x08
#define DARKGREY      DARKGRAY
#define LIGHTBLUE     0x09
#define LIGHTGREEN    0x0a
#define LIGHTCYAN     0x0b
#define LIGHTRED      0x0c
#define LIGHTMAGENTA  0x0d
#define LIGHTPURPLE   LIGHTMAGENTA
#define YELLOW        0x0e
#define WHITE         0x0f
#define BLINK         0x80

#define CURSORHIDE    0
#define CURSORNORM    1
#define CURSORHALF    2
#define CURSORFULL    3

#define vm_mkcolor(fore, back)  ((fore) | (back << 4))
#define vm_fore(attr)           (attr % 16)
#define vm_back(attr)           (attr / 16)
#define vm_mkcolour             vm_mkcolor
#define vm_attrib               vm_paintbox

struct vm_info
{
    char attr;                  /* text attribute */
#if defined(DOS)
    char mode;                  /* video mode */
#endif
    char height;                /* screen height */
    char width;                 /* screen width */
```

```
101 |     char ypos;                  /* y-coordinate */
102 |     char xpos;                  /* x-coordinate */
103 | #if defined(WINNT)
104 |     unsigned short dwSize;      /* 0=minimal; 49=half; 99=full */
105 |     int bVisible;               /* 0=cursor invisible */
106 | #else
107 |     char cur_start;             /* cursor start line */
108 |     char cur_end;               /* cursor end line */
109 | #endif
110 | };
111 |
112 | extern struct vm_info vm_startup;
113 | extern char vm_curattr;
114 |
115 | extern char vm_frame_blank[];
116 | extern char vm_frame_single[];
117 | extern char vm_frame_double[];
118 |
119 | void vm_init(void);
120 | void vm_done(void);
121 | char vm_getscreenwidth(void);
122 | char vm_getscreenheight(void);
123 | short vm_getscreensize(void);
124 | void vm_gotoxy(char x, char y);
125 | char vm_wherex(void);
126 | char vm_wherey(void);
127 | int vm_kbhit(void);
128 | int vm_getch(void);
129 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr);
130 | void vm_paintclearscreen(char attr);
131 | void vm_paintclearline(char row, char attr);
132 | void vm_paintcleareol(char row, char attr);
133 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr);
134 | void vm_paintscreen(char attr);
135 | void vm_paintline(char row, char attr);
136 | void vm_painteol(char row, char attr);
137 | void vm_clearbox(char x1, char y1, char x2, char y2);
138 | void vm_clearscreen(void);
139 | void vm_clearline(char row);
140 | void vm_cleareol(char row);
141 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch);
142 | void vm_fillscreen(char ch);
143 | void vm_fillline(char row, char ch);
144 | void vm_filleol(char row, char ch);
145 | void vm_clrscr(void);
146 | void vm_clreol(void);
147 | char vm_getchxy(char x, char y);
148 | char vm_getattrxy(char x, char y);
149 | void vm_xgetchxy(char x, char y, char *attr, char *ch);
150 | void vm_setattr(char attr);
151 | void vm_putattr(char x, char y, char attr);
152 | void vm_setcursorstyle(int style);
153 | void vm_putch(char x, char y, char ch);
154 | void vm_puts(char x, char y, char *str);
155 | void vm_printf(char x, char y, const char *format,...);
156 | void vm_printfcenter(char row, const char *format,...);
157 | void vm_printfbetween(char x1, char x2, char row, const char *format,...);
158 | void vm_xputch(char x, char y, char attr, char ch);
159 | void vm_xputs(char x, char y, char attr, char *str);
160 | void vm_xprintf(char x, char y, char attr, const char *format,...);
161 | void vm_xprintfcenter(char row, char attr, const char *format,...);
162 | void vm_xprintfbetween(char x1, char x2, char row, char attr, const char *format,...);
163 | void vm_horizline(char x1, char x2, char row, char attr, char ch);
164 | void vm_vertline(char y1, char y2, char col, char attr, char ch);
165 | void vm_frame(char x1, char y1, char x2, char y2, char attr, char *frame);
166 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest);
167 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce);
168 | void vm_getinfo(struct vm_info *v);
169 |
170 | #endif  /* __VIDMGR_H__ */
```

TEXT STATISTICS

5715 characters
170 lines

LEXICAL STATISTICS

13 comments [std-C]
0 comments [C++]
58 preprocessor instructions
0 constants [character]
0 constants [string]
24 constants [numeric]

SYMBOL TABLE

BLACK 60
BLINK 80
BLUE 61
BROWN 67
CURSORFULL 85
CURSORHALF 84
CURSORHIDE 82
CURSORNORM 83
CYAN 63
DARKGRAY 70 71
DARKGREY 71
DOS 36 38 96
EMX 48 50
GREEN 62
LIGHTBLUE 72
LIGHTCYAN 74
LIGHTGRAY 68 69
LIGHTGREEN 73
LIGHTGREY 69
LIGHTMAGENTA 76 77
LIGHTPURPLE 77
LIGHTRED 75
MAGENTA 65 66
MSDOS 37
OS2 42 44
OS_2 43
PURPLE 66
RED 64
WHITE 79
WINNT 54 56 103
YELLOW 78
_QC 37
__DOS__ 37
__EMX__ 49
__MSDOS__ 37
__NT__ 55
__OS2__ 43
__VIDMGR_H__ 33 34
attr 88+ 89+ 95 129 130 131 132 133 134 135 136
149 150 151 158 159 160 161 162 163 164 165
bVisible 105
back 87+
ch 141 142 143 144 149 153 158 163 164
col 164
cur_end 108
cur_start 107
defined 37+ 43+ 49 55 96 103
dest 166
dwSize 104
fore 87+
format 155 156 157 160 161 162
frame 165
height 99
mode 97
row 131 132 135 136 139 140 143 144 156 157 161
162 163
srce 167
str 154 159
style 152
v 168
vm_attrib 91
vm_back 89
vm_clearbox 137
vm_cleareol 140
vm_clearline 139
vm_clearscreen 138
vm_clreol 146
vm_clrscr 145
vm_curattr 113
vm_done 120
vm_fillbox 141
vm_filleol 144
vm_fillline 143
vm_fillscreen 142
vm_fore 88
vm_frame 165
vm_frame_blank 115
vm_frame_double 117
vm_frame_single 116
vm_getattrxy 148
vm_getch 128
vm_getchxy 147
vm_getinfo 168
vm_getscreenheight 122

```
vm_getscreensize        123
vm_getscreenwidth       121
vm_gettext              166
vm_gotoxy       124
vm_horizline            163
vm_info          93     112     168
vm_init         119
vm_kbhit        127
vm_mkcolor               87      90
vm_mkcolour              90
vm_paintbox              91     133
vm_paintclearbox        129
vm_paintcleareol        132
vm_paintclearline       131
vm_paintclearscreen             130
vm_painteol             136
vm_paintline            135
vm_paintscreen          134
vm_printf       155
vm_printfbetween        157
vm_printfcenter         156
vm_putattr              151
vm_putch        153
vm_puts         154
vm_puttext              167
vm_setattr              150
vm_setcursorstyle       152
vm_startup              112
vm_vertline             164
vm_wherex       125
vm_wherey       126
vm_xgetchxy             149
vm_xprintf              160
vm_xprintfbetween       162
vm_xprintfcenter        161
vm_xputch       158
vm_xputs        159
width           100
x       124     147     148     149     151     153     154     155     158     159     160
x1              129     133     137     141     157     162     163     165     166     167
x2              129     133     137     141     157     162     163     165     166     167
xpos            102
y       124     147     148     149     151     153     154     155     158     159     160
y1              129     133     137     141     164     165     166     167
y2              129     133     137     141     164     165     166     167
ypos            101
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  Portable PC screen functions
**  Public domain by Bob Stout
**  Uses SCROLL.C, also from SNIPPETS
*/

#include <stdio.h>
#include <dos.h>
#include "scrnmacs.h"         /* Also in SNIPPETS     */
#include "mk_fp.h"

void GotoXY(int col, int row)
{
      union REGS regs;

      setbuf(stdout, NULL);
      regs.h.dh = (unsigned char)row;
      regs.h.dl = (unsigned char)col;
      regs.h.bh = VIDPAGE;
      regs.h.ah = 2;
      int86(0x10, &regs, &regs);
}

void ClrScrn(int vattrib)
{
      scroll(SCROLL_UP, 0, vattrib, 0, 0, SCREENROWS, SCREENCOLS);
      GotoXY(0, 0);                     /* Home cursor  */
}

void GetCurPos(int *col, int *row)
{
      union REGS regs;

      regs.h.ah = 0x03;
      regs.h.bh = VIDPAGE;
      int86(0x10, &regs, &regs);
      *row = regs.h.dh;
      *col = regs.h.dl;
}

int GetCurAtr(void)
{
      int row, col;
      unsigned short chat;

      GetCurPos(&col, &row);
      chat = *((unsigned short FAR *)MK_FP(SCREENSEG,
            (row * SCREENCOLS + col) << 1));
      return (chat >> 8);
}

void ClrEol(void)
{
      int row, col;

      GetCurPos(&col, &row);
      scroll(0, 0, GetCurAtr(), row, col, row, SCREENCOLS);
}

void ClrEop(void)
{
      int row, col;

      GetCurPos(&col, &row);
      ClrEol();
      if (++row < SCREENROWS)
            scroll(0, 0, GetCurAtr(), row, 0, SCREENROWS, SCREENCOLS);
}

void Repaint(int vattrib)
{
      unsigned short FAR *screen = (unsigned short FAR *)SCRBUFF;
      int row, col;

      for (row = 0; row < SCREENROWS; ++row)
      {
            for (col = 0; col < SCREENCOLS; ++col, ++screen)
                  *screen = (*screen & 0xff) + (vattrib << 8);
      }
}


#ifdef TEST

#include <conio.h>

/*
**  Run this test with a screenful of misc. stuff
*/

main()
{
      int vatr = GetCurAtr();

      GotoXY(1, 1);
      fputs("Testing ClrEol()", stderr);
      ClrEol();
      fputs("\nHit any key to continue...\n", stderr);
```

```
 101 |       getch();
 102 |       fputs("Testing ClrEop()", stderr);
 103 |       ClrEop();
 104 |       fputs("\nHit any key to continue...\n", stderr);
 105 |       getch();
 106 |       ClrScrn(vatr);
 107 |       GotoXY(0, 0);
 108 |       fputs("ClrScrn() tested", stderr);
 109 |       fputs("\nHit any key to continue...\n", stderr);
 110 |       getch();
 111 |       Repaint(BG_(CYAN_) | BLACK_);
 112 |       fputs("Repaint() tested", stderr);
 113 |       fputs("\nHit any key to continue...\n", stderr);
 114 |       getch();
 115 |       Repaint(vatr);
 116 |       return 0;
 117 | }
 118 |
 119 | #endif /* TEST */
```

TEXT STATISTICS

```
  2596 characters
   119 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     7 preprocessor instructions
     0 constants [character]
    10 constants [string]
    25 constants [numeric]
```

SYMBOL TABLE

```
    BG_             111
    BLACK_          111
    CYAN_           111
    ClrEol           54      67      99
    ClrEop           62     103
    ClrScrn          26     106
    FAR              49      74+
    GetCurAtr        43      59      69      95
    GetCurPos        32      48      58      66
    GotoXY           14      29      97     107
    MK_FP            49
    NULL             18
    REGS             16      34
    Repaint          72     111     115
    SCRBUFF          74
    SCREENCOLS               28      50      59      69      79
    SCREENROWS               28      68      69      77
    SCREENSEG        49
    SCROLL_UP        28
    TEST             85
    VIDPAGE          21      37
    ah               22      36
    bh               21      37
    chat             46      49      51
    col              14      20      32      40      45      48      50      56      58      59      64
            66       75      79+
    conio            87
    dh               19      39
    dl               20      40
    dos              10
    fputs            98     100     102     104     108     109     112     113
    getch           101     105     110     114
    h          9     10      19      20      21      22      36      37      39      40      87
    int86            23      38
    main             93
    regs             16      19      20      21      22      23+     34      36      37      38+     39
            40
    row              14      19      32      39      45      48      50      56      58      59+     64
            66       68      69      75      77+
    screen           74      79      80+
    scroll           28      59      69
    setbuf           18
    stderr           98     100     102     104     108     109     112     113
    stdio             9
    stdout           18
    vatr             95     106     115
    vattrib          26      28      72      80
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VIOIMAGE.C; VidMgr routines for saving and restoring text images.
 *              Release 1.2.
 *
 *  This module written in May 1996 by Andrew Clarke and released to the
 *  public domain.
 */

#include <stdlib.h>
#include "vidmgr.h"
#include "vioimage.h"

void vioImageDefaults(VIOIMAGE * v)
{
    v->width = 0;
    v->height = 0;
    v->image = NULL;
}

int vioImageInit(VIOIMAGE * v, char width, char height)
{
    v->image = malloc(width * height * 2);
    if (v->image)
    {
        v->width = width;
        v->height = height;
        return 1;
    }
    else
    {
        return 0;
    }
}

int vioImageTerm(VIOIMAGE * v)
{
    if (v->image)
    {
        free(v->image);
        return 1;
    }
    else
    {
        return 0;
    }
}

int vioImageSave(VIOIMAGE * v, char x, char y)
{
    if (v->image)
    {
        vm_gettext(x, y, v->width, v->height, v->image);
        return 1;
    }
    else
    {
        return 0;
    }
}

int vioImageRestore(VIOIMAGE * v, char x, char y)
{
    if (v->image)
    {
        vm_puttext(x, y, v->width, v->height, v->image);
        return 1;
    }
    else
    {
        return 0;
    }
}
```

TEXT STATISTICS

```
  1266 characters
    74 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     2 constants [string]
    11 constants [numeric]
```

SYMBOL TABLE

```
    NULL            19
    VIOIMAGE        15      22      37      50      63
    free            41
    h       11
    height          18      22      24      28+     54      67
    image           19      24      25      39      41      52      54      65      67
    malloc          24
    stdlib          11
    v       15      17      18      19      22      24      25      27      28      37      39      41
            50      52      54+     63      65      67+
    vioImageDefaults        15
    vioImageInit            22
    vioImageRestore         63
    vioImageSave            50
    vioImageTerm            37
    vm_gettext              54
    vm_puttext              67
    width           17      22      24      27+     54      67
    x       50      54      63      67
    y       50      54      63      67
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 |  *  VIOIMAGE.H; VidMgr routines for saving and restoring text images.
  5 |  *              Release 1.2.
  6 |  *
  7 |  *  This module written in May 1996 by Andrew Clarke and released to the
  8 |  *  public domain.
  9 |  */
 10 |
 11 | #ifndef __VIOIMAGE_H__
 12 | #define __VIOIMAGE_H__
 13 |
 14 | typedef struct
 15 | {
 16 |     char *image;
 17 |     char width;
 18 |     char height;
 19 | }
 20 | VIOIMAGE;
 21 |
 22 | void vioImageDefaults(VIOIMAGE * v);
 23 | int vioImageInit(VIOIMAGE * v, char width, char height);
 24 | int vioImageTerm(VIOIMAGE * v);
 25 | int vioImageSave(VIOIMAGE * v, char x, char y);
 26 | int vioImageRestore(VIOIMAGE * v, char x, char y);
 27 |
 28 | #endif
```

TEXT STATISTICS

```
   632 characters
    28 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    VIOIMAGE          20      22      23      24      25      26
    __VIOIMAGE_H__            11      12
    height            18      23
    image             16
    v         22      23      24      25      26
    vioImageDefaults          22
    vioImageInit              23
    vioImageRestore           26
    vioImageSave              25
    vioImageTerm              24
    width             17      23
    x         25      26
    y         25      26
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VMGRDJGP.C; VidMgr module for the DJGPP GNU C/C++ compiler.  Release 1.2.
 *
 *  This module written in May 1996 by Andrew Clarke and released to the
 *  public domain.
 */

#include <dos.h>
#include <go32.h>
#include <sys/farptr.h>
#include "vidmgr.h"
#include "opsys.h"

static int vm_iscolorcard(void);
static char vm_getscreenmode(void);
static void vm_setscreenmode(char mode);
static void vm_setcursorsize(char start, char end);
static void vm_getcursorsize(char *start, char *end);
static void vm_getkey(unsigned char *chScan, unsigned char *chChar);
static unsigned long vm_screenaddress(char x, char y);

void vm_init(void)
{
    vm_getinfo(&vm_startup);
    vm_setattr(vm_startup.attr);
    opsysDetect();
}

void vm_done(void)
{
    if (vm_getscreenmode() != vm_startup.mode)
    {
        vm_setscreenmode(vm_startup.mode);
    }
    vm_setcursorsize(vm_startup.cur_start, vm_startup.cur_end);
}

void vm_getinfo(struct vm_info *v)
{
    v->ypos = vm_wherey();
    v->xpos = vm_wherex();
    v->attr = vm_getattrxy(v->xpos, v->ypos);
    v->mode = vm_getscreenmode();
    v->height = vm_getscreenheight();
    v->width = vm_getscreenwidth();
    vm_getcursorsize(&v->cur_start, &v->cur_end);
}

static int vm_iscolorcard(void)
{
    return vm_getscreenmode() != 7;
}

static char vm_getscreenmode(void)
{
    return (char)_farpeekb(_go32_conventional_mem_selector(), 0x0449);
}

static void vm_setscreenmode(char mode)
{
    union REGS regs;
    regs.h.ah = 0x00;
    regs.h.al = mode;
    int86(0x10, &regs, &regs);
}

char vm_getscreenwidth(void)
{
    return (char)_farpeekb(_go32_conventional_mem_selector(), 0x044a);
}

char vm_getscreenheight(void)
{
    return (char)(_farpeekb(_go32_conventional_mem_selector(), 0x0484) + 1);
}

short vm_getscreensize(void)
{
    return (short)_farpeekw(_go32_conventional_mem_selector(), 0x044c);
}

void vm_gotoxy(char x, char y)
{
    union REGS regs;
    regs.h.ah = 0x02;
    regs.h.bh = 0;
    regs.h.dh = (unsigned char)(y - 1);
    regs.h.dl = (unsigned char)(x - 1);
    int86(0x10, &regs, &regs);
}

char vm_wherex(void)
{
    return (char)(_farpeekb(_go32_conventional_mem_selector(), 0x0450) + 1);
}

char vm_wherey(void)
{
```

```
101 |     return (char)(_farpeekb(_go32_conventional_mem_selector(), 0x0451) + 1);
102 | }
103 |
104 | static void vm_setcursorsize(char start, char end)
105 | {
106 |     union REGS regs;
107 |     regs.h.ah = 0x01;
108 |     regs.h.ch = start;
109 |     regs.h.cl = end;
110 |     int86(0x10, &regs, &regs);
111 | }
112 |
113 | static void vm_getcursorsize(char *start, char *end)
114 | {
115 |     union REGS regs;
116 |     regs.h.ah = 0x03;
117 |     regs.h.bh = 0;
118 |     int86(0x10, &regs, &regs);
119 |     *start = regs.h.ch;
120 |     *end = regs.h.cl;
121 | }
122 |
123 | int vm_kbhit(void)
124 | {
125 |     union REGS regs;
126 |     static unsigned short counter = 0;
127 |     if (counter % 10 == 0)
128 |     {
129 |         opsysTimeSlice();
130 |     }
131 |     counter++;
132 |     regs.h.ah = 0x01;
133 |     int86(0x16, &regs, &regs);
134 |     return !(regs.x.flags & 0x40);
135 | }
136 |
137 | static void vm_getkey(unsigned char *chScan, unsigned char *chChar)
138 | {
139 |     union REGS regs;
140 |     regs.h.ah = 0x00;
141 |     int86(0x16, &regs, &regs);
142 |     *chScan = regs.h.ah;
143 |     *chChar = regs.h.al;
144 | }
145 |
146 | int vm_getch(void)
147 | {
148 |     unsigned char chChar, chScan;
149 |
150 |     while (!vm_kbhit())
151 |     {
152 |         /* nada */
153 |     }
154 |
155 |     vm_getkey(&chScan, &chChar);
156 |     if (chChar == 0xe0)
157 |     {
158 |         if (chScan)
159 |         {
160 |             chChar = 0;         /* force scan return */
161 |         }
162 |         else
163 |         {                       /* get next block */
164 |             chChar = 0;
165 |
166 |             vm_getkey(&chScan, &chChar);
167 |             if (!chScan)
168 |             {                   /* still no scan? */
169 |                 chScan = chChar;  /* move new char over */
170 |                 chChar = 0;     /* force its return */
171 |             }
172 |             else
173 |             {
174 |                 chChar = 0;     /* force new scan */
175 |             }
176 |         }
177 |     }
178 |     if (chScan == 0xe0)
179 |     {
180 |         if (!chChar)
181 |         {
182 |             chScan = 0;
183 |
184 |             vm_getkey(&chScan, &chChar);
185 |             if (!chScan)
186 |             {                   /* still no scan? */
187 |                 chScan = chChar;  /* move new char over */
188 |                 chChar = 0;     /* force its return */
189 |             }
190 |             else
191 |             {
192 |                 chChar = 0;     /* force new scan */
193 |             }
194 |         }
195 |         else
196 |         {
197 |             chScan = 0;         /* handle 0xe00d case */
198 |         }
199 |     }
200 |     if (chChar)
```

```
201 |     {
202 |         chScan = 0;
203 |     }
204 |
205 |     return (int)((chScan << 8) + (chChar));
206 | }
207 |
208 | void vm_setcursorstyle(int style)
209 | {
210 |     if (vm_iscolorcard())
211 |     {
212 |         switch (style)
213 |         {
214 |         case CURSORHALF:
215 |             vm_setcursorsize(4, 7);
216 |             break;
217 |         case CURSORFULL:
218 |             vm_setcursorsize(0, 7);
219 |             break;
220 |         case CURSORNORM:
221 |             vm_setcursorsize(7, 8);
222 |             break;
223 |         case CURSORHIDE:
224 |             vm_setcursorsize(32, 32);
225 |             break;
226 |         default:
227 |             break;
228 |         }
229 |     }
230 |     else
231 |     {
232 |         switch (style)
233 |         {
234 |         case CURSORHALF:
235 |             vm_setcursorsize(8, 13);
236 |             break;
237 |         case CURSORFULL:
238 |             vm_setcursorsize(0, 13);
239 |             break;
240 |         case CURSORNORM:
241 |             vm_setcursorsize(11, 13);
242 |             break;
243 |         case CURSORHIDE:
244 |             vm_setcursorsize(32, 32);
245 |             break;
246 |         default:
247 |             break;
248 |         }
249 |     }
250 | }
251 |
252 | static unsigned long vm_screenaddress(char x, char y)
253 | {
254 |     unsigned short base;
255 |     if (vm_iscolorcard())
256 |     {
257 |         base = opsysGetVideoSeg(0xB800);
258 |     }
259 |     else
260 |     {
261 |         base = opsysGetVideoSeg(0xB000);
262 |     }
263 |     return (unsigned long)((base << 4) + ((y - 1) * vm_getscreenwidth() * 2) + ((x - 1) * 2));
264 | }
265 |
266 | char vm_getchxy(char x, char y)
267 | {
268 |     unsigned long address;
269 |     address = vm_screenaddress(x, y);
270 |     return (char)_farpeekb(_go32_conventional_mem_selector(), address);
271 | }
272 |
273 | char vm_getattrxy(char x, char y)
274 | {
275 |     unsigned long address;
276 |     address = vm_screenaddress(x, y);
277 |     return (char)_farpeekb(_go32_conventional_mem_selector(), address + 1L);
278 | }
279 |
280 | void vm_xgetchxy(char x, char y, char *attr, char *ch)
281 | {
282 |     unsigned long address;
283 |     address = vm_screenaddress(x, y);
284 |     *ch = (char)_farpeekb(_go32_conventional_mem_selector(), address);
285 |     *attr = (char)_farpeekb(_go32_conventional_mem_selector(), address + 1L);
286 | }
287 |
288 | void vm_putch(char x, char y, char ch)
289 | {
290 |     unsigned long address;
291 |     address = vm_screenaddress(x, y);
292 |     _farpokeb(_go32_conventional_mem_selector(), address, ch);
293 | }
294 |
295 | void vm_puts(char x, char y, char *str)
296 | {
297 |     char ofs;
298 |     ofs = 0;
299 |     while (*str)
300 |     {
```

```
301 |          unsigned long address;
302 |          address = vm_screenaddress(x + ofs, y);
303 |          _farpokeb(_go32_conventional_mem_selector(), address, *str);
304 |          str++;
305 |          ofs++;
306 |      }
307 | }
308 |
309 | void vm_xputch(char x, char y, char attr, char ch)
310 | {
311 |     unsigned long address;
312 |     address = vm_screenaddress(x, y);
313 |     _farpokeb(_go32_conventional_mem_selector(), address, ch);
314 |     _farpokeb(_go32_conventional_mem_selector(), address + 1L, attr);
315 | }
316 |
317 | void vm_xputs(char x, char y, char attr, char *str)
318 | {
319 |     char ofs;
320 |     ofs = 0;
321 |     while (*str)
322 |     {
323 |         unsigned long address;
324 |         address = vm_screenaddress(x + ofs, y);
325 |         _farnspokeb(address, *str);
326 |         _farnspokeb(address + 1L, attr);
327 |         str++;
328 |         ofs++;
329 |     }
330 | }
331 |
332 | void vm_putattr(char x, char y, char attr)
333 | {
334 |     unsigned long address;
335 |     address = vm_screenaddress(x, y);
336 |     _farpokeb(_go32_conventional_mem_selector(), address + 1L, attr);
337 | }
338 |
339 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr)
340 | {
341 |     char x, y;
342 |     for (y = y1; y <= y2; y++)
343 |     {
344 |         for (x = x1; x <= x2; x++)
345 |         {
346 |             vm_xputch(x, y, attr, ' ');
347 |         }
348 |     }
349 | }
350 |
351 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr)
352 | {
353 |     char x, y;
354 |     for (y = y1; y <= y2; y++)
355 |     {
356 |         for (x = x1; x <= x2; x++)
357 |         {
358 |             vm_putattr(x, y, attr);
359 |         }
360 |     }
361 | }
362 |
363 | void vm_clearbox(char x1, char y1, char x2, char y2)
364 | {
365 |     char x, y;
366 |     for (y = y1; y <= y2; y++)
367 |     {
368 |         for (x = x1; x <= x2; x++)
369 |         {
370 |             vm_putch(x, y, ' ');
371 |         }
372 |     }
373 | }
374 |
375 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch)
376 | {
377 |     char x, y;
378 |     for (y = y1; y <= y2; y++)
379 |     {
380 |         for (x = x1; x <= x2; x++)
381 |         {
382 |             vm_putch(x, y, ch);
383 |         }
384 |     }
385 | }
386 |
387 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest)
388 | {
389 |     char x, y;
390 |     for (y = y1; y <= y2; y++)
391 |     {
392 |         for (x = x1; x <= x2; x++)
393 |         {
394 |             vm_xgetchxy(x, y, dest + 1, dest);
395 |             dest += 2;
396 |         }
397 |     }
398 | }
399 |
400 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce)
```

```
401 | {
402 |     char x, y;
403 |     for (y = y1; y <= y2; y++)
404 |     {
405 |         for (x = x1; x <= x2; x++)
406 |         {
407 |             vm_xputch(x, y, *(srce + 1), *srce);
408 |             srce += 2;
409 |         }
410 |     }
411 | }
412 |
413 | void vm_horizline(char x1, char x2, char row, char attr, char ch)
414 | {
415 |     char x;
416 |     for (x = x1; x <= x2; x++)
417 |     {
418 |         vm_xputch(x, row, attr, ch);
419 |     }
420 | }
```

TEXT STATISTICS

9483 characters
420 lines

LEXICAL STATISTICS

14 comments [std-C]
0 comments [C++]
5 preprocessor instructions
2 constants [character]
2 constants [string]
76 constants [numeric]

SYMBOL TABLE

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VMGRDOS.C  VidMgr module for MS-DOS compilers.  Release 1.3.
 *
 *  This module written in March 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in October 1996.
 */

#include <string.h>
#include <dos.h>
#include "vidmgr.h"
#include "opsys.h"

#if defined(__POWERC) || (defined(__TURBOC__) && !defined(__BORLANDC__)) || \
  (defined(__ZTC__) && !defined(__SC__))
#define FAR far
#else
#if defined(__MSDOS__) || defined(MSDOS) || defined(DOS)
#define FAR _far
#else
#define FAR
#endif
#endif

#ifndef MK_FP
#define MK_FP(seg,off) \
  ((void FAR *)(((unsigned long)(seg) << 16)|(unsigned)(off)))
#endif

static int vm_iscolorcard(void);
static char vm_getscreenmode(void);
static void vm_setscreenmode(char mode);
static void vm_setcursorsize(char start, char end);
static void vm_getcursorsize(char *start, char *end);
static void vm_getkey(unsigned char *chScan, unsigned char *chChar);

void vm_init(void)
{
    vm_getinfo(&vm_startup);
    vm_setattr(vm_startup.attr);
    opsysDetect();
}

void vm_done(void)
{
    if (vm_getscreenmode() != vm_startup.mode)
    {
        vm_setscreenmode(vm_startup.mode);
    }
    vm_setcursorsize(vm_startup.cur_start, vm_startup.cur_end);
}

void vm_getinfo(struct vm_info *v)
{
    v->ypos = vm_wherey();
    v->xpos = vm_wherex();
    v->attr = vm_getattrxy(v->xpos, v->ypos);
    v->mode = vm_getscreenmode();
    v->height = vm_getscreenheight();
    v->width = vm_getscreenwidth();
    vm_getcursorsize(&v->cur_start, &v->cur_end);
}

static int vm_iscolorcard(void)
{
    return vm_getscreenmode() != 7;
}

#if defined(__WATCOMC__) && defined(__386__)

char *vm_screenptr(char x, char y)
{
    char *ptr;
    if (vm_iscolorcard())
    {
        ptr = (char *)(opsysGetVideoSeg(0xB800) << 4);
    }
    else
    {
        ptr = (char *)(opsysGetVideoSeg(0xB000) << 4);
    }
    return ptr + (y * vm_getscreenwidth() * 2) + (x * 2);
}

#else

char FAR *vm_screenptr(char x, char y)
{
    char FAR *ptr;
    if (vm_iscolorcard())
    {
        ptr = (char FAR *)MK_FP(opsysGetVideoSeg(0xB800), 0x0000);
    }
    else
    {
        ptr = (char FAR *)MK_FP(opsysGetVideoSeg(0xB000), 0x0000);
    }
    return ptr + (y * vm_getscreenwidth() * 2) + (x * 2);
}
```

```
101 |
102 | #endif
103 |
104 | static char vm_getscreenmode(void)
105 | {
106 | #if defined(__WATCOMC__) && defined(__386__)
107 |     return *((char *)0x0449);
108 | #else
109 |     return *((char FAR *)MK_FP(0x0040, 0x0049));
110 | #endif
111 | }
112 |
113 | static void vm_setscreenmode(char mode)
114 | {
115 | #if defined(__TURBOC__)
116 |     _AH = 0x00;
117 |     _AL = mode;
118 |     geninterrupt(0x10);
119 | #else
120 |     union REGS regs;
121 |     regs.h.ah = 0x00;
122 |     regs.h.al = mode;
123 | #if defined(__WATCOMC__) && defined(__386__)
124 |     int386(0x10, &regs, &regs);
125 | #else
126 |     int86(0x10, &regs, &regs);
127 | #endif
128 | #endif
129 | }
130 |
131 | char vm_getscreenwidth(void)
132 | {
133 | #if defined(__WATCOMC__) && defined(__386__)
134 |     return *((char *)0x044a);
135 | #else
136 |     return *((char FAR *)MK_FP(0x0040, 0x004a));
137 | #endif
138 | }
139 |
140 | char vm_getscreenheight(void)
141 | {
142 | #if defined(__WATCOMC__) && defined(__386__)
143 |     return (char)(*((char *)0x0484) + 1);
144 | #else
145 |     return (char)(*((char FAR *)MK_FP(0x0040, 0x0084)) + 1);
146 | #endif
147 | }
148 |
149 | short vm_getscreensize(void)
150 | {
151 | #if defined(__WATCOMC__) && defined(__386__)
152 |     return *((short *)0x044c);
153 | #else
154 |     return *((short FAR *)MK_FP(0x0040, 0x004c));
155 | #endif
156 | }
157 |
158 | void vm_gotoxy(char x, char y)
159 | {
160 | #if defined(__TURBOC__)
161 |     _AH = 0x02;
162 |     _BH = 0;
163 |     _DH = y - 1;
164 |     _DL = x - 1;
165 |     geninterrupt(0x10);
166 | #else
167 |     union REGS regs;
168 |     regs.h.ah = 0x02;
169 |     regs.h.bh = 0;
170 |     regs.h.dh = (unsigned char)(y - 1);
171 |     regs.h.dl = (unsigned char)(x - 1);
172 | #if defined(__WATCOMC__) && defined(__386__)
173 |     int386(0x10, &regs, &regs);
174 | #else
175 |     int86(0x10, &regs, &regs);
176 | #endif
177 | #endif
178 | }
179 |
180 | char vm_wherex(void)
181 | {
182 | #if defined(__WATCOMC__) && defined(__386__)
183 |     return (char)(*((char *)0x0450) + 1);
184 | #else
185 |     return (char)(*((char FAR *)MK_FP(0x0040, 0x0050)) + 1);
186 | #endif
187 | }
188 |
189 | char vm_wherey(void)
190 | {
191 | #if defined(__WATCOMC__) && defined(__386__)
192 |     return (char)(*((char *)0x0451) + 1);
193 | #else
194 |     return (char)(*((char FAR *)MK_FP(0x0040, 0x0051)) + 1);
195 | #endif
196 | }
197 |
198 | static void vm_setcursorsize(char start, char end)
199 | {
200 | #if defined(__TURBOC__)
```

```
201 |     _AH = 0x01;
202 |     _CH = start;
203 |     _CL = end;
204 |     geninterrupt(0x10);
205 | #else
206 |     union REGS regs;
207 |     regs.h.ah = 0x01;
208 |     regs.h.ch = start;
209 |     regs.h.cl = end;
210 | #if defined(__WATCOMC__) && defined(__386__)
211 |     int386(0x10, &regs, &regs);
212 | #else
213 |     int86(0x10, &regs, &regs);
214 | #endif
215 | #endif
216 | }
217 |
218 | static void vm_getcursorsize(char *start, char *end)
219 | {
220 | #if defined(__TURBOC__)
221 |     _AH = 0x03;
222 |     _BH = 0;
223 |     geninterrupt(0x10);
224 |     *start = _CH;
225 |     *end = _CL;
226 | #else
227 |     union REGS regs;
228 |     regs.h.ah = 0x03;
229 |     regs.h.bh = 0;
230 | #if defined(__WATCOMC__) && defined(__386__)
231 |     int386(0x10, &regs, &regs);
232 | #else
233 |     int86(0x10, &regs, &regs);
234 | #endif
235 |     *start = regs.h.ch;
236 |     *end = regs.h.cl;
237 | #endif
238 | }
239 |
240 | int vm_kbhit(void)
241 | {
242 |     static unsigned short counter = 0;
243 |     if (counter % 10 == 0)
244 |     {
245 |         opsysTimeSlice();
246 |     }
247 |     counter++;
248 | #if defined(_MSC_VER) || defined(_QC) || defined(__SC__)
249 |     _asm
250 |     {
251 |         mov    ah,0x01
252 |         int    16h
253 |         jz     nokey
254 |     }
255 |     return 1;
256 | nokey:
257 |     return 0;
258 | #elif defined(__TURBOC__)
259 |     _AH = 0x01;
260 |     geninterrupt(0x16);
261 |     return !(_FLAGS & 0x40);
262 | #else
263 |     {
264 |         union REGPACK regpack;
265 |         memset(&regpack, 0, sizeof regpack);
266 |         regpack.h.ah = 0x01;
267 |         intr(0x16, &regpack);
268 |         return !(regpack.x.flags & 0x40);
269 |     }
270 | #endif
271 | }
272 |
273 | static void vm_getkey(unsigned char *chScan, unsigned char *chChar)
274 | {
275 | #if defined(__TURBOC__)
276 |     _AH = 0x00;
277 |     geninterrupt(0x16);
278 |     *chScan = _AH;
279 |     *chChar = _AL;
280 | #else
281 |     union REGS regs;
282 |     regs.h.ah = 0x00;
283 | #if defined(__WATCOMC__) && defined(__386__)
284 |     int386(0x16, &regs, &regs);
285 | #else
286 |     int86(0x16, &regs, &regs);
287 | #endif
288 |     *chScan = regs.h.ah;
289 |     *chChar = regs.h.al;
290 | #endif
291 | }
292 |
293 | int vm_getch(void)
294 | {
295 |     unsigned char chChar, chScan;
296 |
297 |     while (!vm_kbhit())
298 |     {
299 |         /* nada */
300 |     }
```

```
301 |
302 |     vm_getkey(&chScan, &chChar);
303 |     if (chChar == 0xe0)
304 |     {
305 |         if (chScan)
306 |         {
307 |             chChar = 0;         /* force scan return */
308 |         }
309 |         else
310 |         {                       /* get next block */
311 |             chChar = 0;
312 |
313 |             vm_getkey(&chScan, &chChar);
314 |             if (!chScan)
315 |             {                   /* still no scan? */
316 |                 chScan = chChar;  /* move new char over */
317 |                 chChar = 0;     /* force its return */
318 |             }
319 |             else
320 |             {
321 |                 chChar = 0;     /* force new scan */
322 |             }
323 |         }
324 |     }
325 |     if (chScan == 0xe0)
326 |     {
327 |         if (!chChar)
328 |         {
329 |             chScan = 0;
330 |
331 |             vm_getkey(&chScan, &chChar);
332 |             if (!chScan)
333 |             {                   /* still no scan? */
334 |                 chScan = chChar;  /* move new char over */
335 |                 chChar = 0;     /* force its return */
336 |             }
337 |             else
338 |             {
339 |                 chChar = 0;     /* force new scan */
340 |             }
341 |         }
342 |         else
343 |         {
344 |             chScan = 0;         /* handle 0xe00d case */
345 |         }
346 |     }
347 |     if (chChar)
348 |     {
349 |         chScan = 0;
350 |     }
351 |
352 |     return (int)((chScan << 8) + (chChar));
353 | }
354 |
355 | void vm_setcursorstyle(int style)
356 | {
357 |     if (vm_iscolorcard())
358 |     {
359 |         switch (style)
360 |         {
361 |         case CURSORHALF:
362 |             vm_setcursorsize(4, 7);
363 |             break;
364 |         case CURSORFULL:
365 |             vm_setcursorsize(0, 7);
366 |             break;
367 |         case CURSORNORM:
368 |             vm_setcursorsize(7, 8);
369 |             break;
370 |         case CURSORHIDE:
371 |             vm_setcursorsize(32, 32);
372 |             break;
373 |         default:
374 |             break;
375 |         }
376 |     }
377 |     else
378 |     {
379 |         switch (style)
380 |         {
381 |         case CURSORHALF:
382 |             vm_setcursorsize(8, 13);
383 |             break;
384 |         case CURSORFULL:
385 |             vm_setcursorsize(0, 13);
386 |             break;
387 |         case CURSORNORM:
388 |             vm_setcursorsize(11, 13);
389 |             break;
390 |         case CURSORHIDE:
391 |             vm_setcursorsize(32, 32);
392 |             break;
393 |         default:
394 |             break;
395 |         }
396 |     }
397 | }
398 |
399 | char vm_getchxy(char x, char y)
400 | {
```

```
401 |     char FAR *p;
402 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
403 |     return *p;
404 | }
405 |
406 | char vm_getattrxy(char x, char y)
407 | {
408 |     char FAR *p;
409 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
410 |     return *(p + 1);
411 | }
412 |
413 | void vm_xgetchxy(char x, char y, char *attr, char *ch)
414 | {
415 |     char FAR *p;
416 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
417 |     *ch = *p;
418 |     *attr = *(p + 1);
419 | }
420 |
421 | void vm_putch(char x, char y, char ch)
422 | {
423 |     char FAR *p;
424 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
425 |     *p = ch;
426 | }
427 |
428 | void vm_puts(char x, char y, char *str)
429 | {
430 |     char FAR *p;
431 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
432 |     while (*str)
433 |     {
434 |         *p++ = *str++;
435 |         p++;
436 |     }
437 | }
438 |
439 | void vm_xputch(char x, char y, char attr, char ch)
440 | {
441 |     char FAR *p;
442 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
443 |     *p = ch;
444 |     *(p + 1) = attr;
445 | }
446 |
447 | void vm_xputs(char x, char y, char attr, char *str)
448 | {
449 |     char FAR *p;
450 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
451 |     while (*str)
452 |     {
453 |         *p++ = *str++;
454 |         *p++ = attr;
455 |     }
456 | }
457 |
458 | void vm_putattr(char x, char y, char attr)
459 | {
460 |     char FAR *p;
461 |     p = vm_screenptr((char)(x - 1), (char)(y - 1));
462 |     *(p + 1) = attr;
463 | }
464 |
465 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr)
466 | {
467 |     char x, y;
468 |     for (y = y1; y <= y2; y++)
469 |     {
470 |         for (x = x1; x <= x2; x++)
471 |         {
472 |             vm_xputch(x, y, attr, ' ');
473 |         }
474 |     }
475 | }
476 |
477 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr)
478 | {
479 |     char x, y;
480 |     for (y = y1; y <= y2; y++)
481 |     {
482 |         for (x = x1; x <= x2; x++)
483 |         {
484 |             vm_putattr(x, y, attr);
485 |         }
486 |     }
487 | }
488 |
489 | void vm_clearbox(char x1, char y1, char x2, char y2)
490 | {
491 |     char x, y;
492 |     for (y = y1; y <= y2; y++)
493 |     {
494 |         for (x = x1; x <= x2; x++)
495 |         {
496 |             vm_putch(x, y, ' ');
497 |         }
498 |     }
499 | }
500 |
```

```
501 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch)
502 | {
503 |     char x, y;
504 |     for (y = y1; y <= y2; y++)
505 |     {
506 |         for (x = x1; x <= x2; x++)
507 |         {
508 |             vm_putch(x, y, ch);
509 |         }
510 |     }
511 | }
512 |
513 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest)
514 | {
515 |     char x, y;
516 |     for (y = y1; y <= y2; y++)
517 |     {
518 |         for (x = x1; x <= x2; x++)
519 |         {
520 |             vm_xgetchxy(x, y, dest + 1, dest);
521 |             dest += 2;
522 |         }
523 |     }
524 | }
525 |
526 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce)
527 | {
528 |     char x, y;
529 |     for (y = y1; y <= y2; y++)
530 |     {
531 |         for (x = x1; x <= x2; x++)
532 |         {
533 |             vm_xputch(x, y, *(srce + 1), *srce);
534 |             srce += 2;
535 |         }
536 |     }
537 | }
538 |
539 | void vm_horizline(char x1, char x2, char row, char attr, char ch)
540 | {
541 |     char x;
542 |     for (x = x1; x <= x2; x++)
543 |     {
544 |         vm_xputch(x, row, attr, ch);
545 |     }
546 | }
```

TEXT STATISTICS

```
 11469 characters
   546 lines
```

LEXICAL STATISTICS

```
    14 comments [std-C]
     0 comments [C++]
    71 preprocessor instructions
     2 constants [character]
     2 constants [string]
   137 constants [numeric]
```

SYMBOL TABLE

```
    CURSORFULL              364     384
    CURSORHALF              361     381
    CURSORHIDE              370     390
    CURSORNORM              367     387
    DOS              19
    FAR               17      20      22      28      88      90      93      97     109     136     145
           154      185     194     401     408     415     423     430     441     449     460
    MK_FP             26      27      93      97     109     136     145     154     185     194
    MSDOS             19
    REGPACK          264
    REGS             120     167     206     227     281
    _AH              116     161     201     221     259     276     278
    _AL              117     279
    _BH              162     222
    _CH              202     224
    _CL              203     225
    _DH              163
    _DL              164
    _FLAGS           261
    _MSC_VER         248
    _QC              248
    __386__           70     106     123     133     142     151     172     182     191     210     230
           283
    __BORLANDC__              15
    __MSDOS__        19
    __POWERC         15
    __SC__           16     248
    __TURBOC__                15     115     160     200     220     258     275
    __WATCOMC__               70     106     123     133     142     151     172     182     191     210
           230      283
    __ZTC__          16
    _asm             249
    _far              20
    ah               121     168     207     228     251     266     282     288
    al               122     289
    attr              41      58     413     418     439     444     447     454     458     462     465
           472      477     484     539     544
    bh               169     229
    ch               208     235     413     417     421     425     439     443     501     508     539
           544
    chChar            36     273     279     289     295     302     303     307     311     313     316
           317      321     327     331     334     335     339     347     352
    chScan            36     273     278     288     295     302     305     313     314     316     325
           329      331     332     334     344     349     352
    cl               209     236
    counter          242     243     247
    cur_end           51      62
    cur_start         51      62
    defined           15+     16+     19+     70+    106+    115     123+    133+    142+    151+    160
           172+     182+    191+    200     210+    220     230+    248+    258     275     283+
    dest             513     520+    521
    dh               170
    dl               171
    dos               11
    end               34      35     198     203     209     218     225     236
    far               17
    flags            268
    geninterrupt             118     165     204     223     260     277
    h         10      11     121     122     168     169     170     171     207     208     209     228
           229      235     236     266     282     288     289
    height            60
    int386           124     173     211     231     284
    int86            126     175     213     233     286
    intr             267
    jz               253
    memset           265
    mode              33      47      49      59     113     117     122
    mov              251
    nokey            253     256
    off               27      28
    opsysDetect               42
    opsysGetVideoSeg          77      81      93      97
    opsysTimeSlice           245
    p        401     402     403     408     409     410     415     416     417     418     423     424
           425     430     431     434     435     441     442     443     444     449     450     453
           454     460     461     462
    ptr               74      77      81      83      90      93      97      99
    regpack          264     265+    266     267     268
    regs             120     121     122     124+    126+    167     168     169     170     171     173+
           175+     206     207     208     209     211+    213+    227     228     229     231+    233+
           235      236     281     282     284+    286+    288     289
    row              539     544
    seg               27      28
    srce             526     533+    534
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VMGREMX.C; VidMgr module for the EMX GNU C/C++ compiler.  Release 1.2.
 *
 *  This module written in April 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in June 1996.
 */

#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <sys/video.h>

#define INCL_KBD
#define INCL_VIO
#define INCL_DOSPROCESS

#include <os2.h>

#include "vidmgr.h"
#include "opsys.h"

static int vm_iscolorcard(void);
static void vm_setcursorsize(char start, char end);
static void vm_getcursorsize(char *start, char *end);
static void vm_getkey(unsigned char *chScan, unsigned char *chChar);

static int video_init = 0;

#define vi_init()  { if (!video_init) video_init = v_init(); }
#define vi_done()  { video_init = 0; }

void vm_init(void)
{
    vi_init();
    vm_getinfo(&vm_startup);
    vm_setattr(vm_startup.attr);
    if (_osmode == DOS_MODE)
    {
        opsysDetect();
    }
}

void vm_done(void)
{
    vm_setcursorsize(vm_startup.cur_start, vm_startup.cur_end);
    vi_done();
}

void vm_getinfo(struct vm_info *v)
{
    v->ypos = vm_wherey();
    v->xpos = vm_wherex();
    v->attr = vm_getattrxy(v->xpos, v->ypos);
    v->height = vm_getscreenheight();
    v->width = vm_getscreenwidth();
    vm_getcursorsize(&v->cur_start, &v->cur_end);
}

char vm_getscreenwidth(void)
{
    if (_osmode == DOS_MODE)
    {
        int width, height;
        vi_init();
        v_dimen(&width, &height);
        return (char)width;
    }
    else
    {
        VIOMODEINFO vi;
        vi.cb = sizeof(VIOMODEINFO);
        VioGetMode(&vi, 0);
        return vi.col;
    }
}

char vm_getscreenheight(void)
{
    if (_osmode == DOS_MODE)
    {
        int width, height;
        vi_init();
        v_dimen(&width, &height);
        return (char)height;
    }
    else
    {
        VIOMODEINFO vi;
        vi.cb = sizeof(VIOMODEINFO);
        VioGetMode(&vi, 0);
        return vi.row;
    }
}

short vm_getscreensize(void)
{
    return (short)(vm_getscreenwidth() * vm_getscreenheight() * 2);
}
```

```
101 |
102 | char vm_wherex(void)
103 | {
104 |     if (_osmode == DOS_MODE)
105 |     {
106 |         int row, col;
107 |         vi_init();
108 |         v_getxy(&col, &row);
109 |         return (char)(col + 1);
110 |     }
111 |     else
112 |     {
113 |         USHORT row, col;
114 |         VioGetCurPos(&row, &col, 0);
115 |         return (char)(col + 1);
116 |     }
117 | }
118 |
119 | char vm_wherey(void)
120 | {
121 |     if (_osmode == DOS_MODE)
122 |     {
123 |         int row, col;
124 |         vi_init();
125 |         v_getxy(&col, &row);
126 |         return (char)(row + 1);
127 |     }
128 |     else
129 |     {
130 |         USHORT row, col;
131 |         VioGetCurPos(&row, &col, 0);
132 |         return (char)(row + 1);
133 |     }
134 | }
135 |
136 | static int vm_iscolorcard(void)
137 | {
138 |     int hw = v_hardware();
139 |     return hw == V_COLOR_8 || hw == V_COLOR_12 ? 1 : 0;
140 | }
141 |
142 | void vm_gotoxy(char x, char y)
143 | {
144 |     if (_osmode == DOS_MODE)
145 |     {
146 |         vi_init();
147 |         v_gotoxy((int)(x - 1), (int)(y - 1));
148 |     }
149 |     else
150 |     {
151 |         VioSetCurPos((USHORT) (y - 1), (USHORT) (x - 1), 0);
152 |     }
153 | }
154 |
155 | static void vm_setcursorsize(char start, char end)
156 | {
157 |     if (_osmode == DOS_MODE)
158 |     {
159 |         vi_init();
160 |         v_ctype(start, end);
161 |     }
162 |     else
163 |     {
164 |         VIOCURSORINFO vi;
165 |         vi.yStart = start;
166 |         vi.cEnd = end;
167 |         vi.cx = 0;
168 |         vi.attr = 0;
169 |         VioSetCurType(&vi, 0);
170 |     }
171 | }
172 |
173 | static void vm_getcursorsize(char *start, char *end)
174 | {
175 |     if (_osmode == DOS_MODE)
176 |     {
177 |         int cstart, cend;
178 |         vi_init();
179 |         v_getctype(&cstart, &cend);
180 |         *start = (char)cstart;
181 |         *end = (char)cend;
182 |     }
183 |     else
184 |     {
185 |         VIOCURSORINFO vi;
186 |         VioGetCurType(&vi, 0);
187 |         *start = vi.yStart;
188 |         *end = vi.cEnd;
189 |     }
190 | }
191 |
192 | static void vm_getkey(unsigned char *chScan, unsigned char *chChar)
193 | {
194 |     union REGS regs;
195 |     regs.h.ah = 0x00;
196 |     _int86(0x16, &regs, &regs);
197 |     *chScan = regs.h.ah;
198 |     *chChar = regs.h.al;
199 | }
200 |
```

```
201 | int vm_kbhit(void)
202 | {
203 |     if (_osmode == DOS_MODE)
204 |     {
205 |         union REGS regs;
206 |         static unsigned short counter = 0;
207 |         if (counter % 10 == 0)
208 |         {
209 |             opsysTimeSlice();
210 |         }
211 |         counter++;
212 |         regs.h.ah = 0x01;
213 |         _int86(0x16, &regs, &regs);
214 |         return !(regs.x.flags & 0x40);
215 |     }
216 |     else
217 |     {
218 |         KBDKEYINFO ki;
219 |         ki.fbStatus = 0;
220 |         KbdPeek(&ki, 0);
221 |         return ki.fbStatus & KBDTRF_FINAL_CHAR_IN;
222 |     }
223 | }
224 |
225 | int vm_getch(void)
226 | {
227 |     if (_osmode == DOS_MODE)
228 |     {
229 |         unsigned char chChar, chScan;
230 |
231 |         while (!vm_kbhit())
232 |         {
233 |             /* nada */
234 |         }
235 |
236 |         vm_getkey(&chScan, &chChar);
237 |         if (chChar == 0xe0)
238 |         {
239 |             if (chScan)
240 |             {
241 |                 chChar = 0;     /* force scan return */
242 |             }
243 |             else
244 |             {                   /* get next block */
245 |                 chChar = 0;
246 |
247 |                 vm_getkey(&chScan, &chChar);
248 |                 if (!chScan)
249 |                 {               /* still no scan? */
250 |                     chScan = chChar;  /* move new char over */
251 |                     chChar = 0; /* force its return */
252 |                 }
253 |                 else
254 |                 {
255 |                     chChar = 0; /* force new scan */
256 |                 }
257 |             }
258 |         }
259 |         if (chScan == 0xe0)
260 |         {
261 |             if (!chChar)
262 |             {
263 |                 chScan = 0;
264 |
265 |                 vm_getkey(&chScan, &chChar);
266 |                 if (!chScan)
267 |                 {               /* still no scan? */
268 |                     chScan = chChar;  /* move new char over */
269 |                     chChar = 0; /* force its return */
270 |                 }
271 |                 else
272 |                 {
273 |                     chChar = 0; /* force new scan */
274 |                 }
275 |             }
276 |             else
277 |             {
278 |                 chScan = 0;     /* handle 0xe00d case */
279 |             }
280 |         }
281 |         if (chChar)
282 |         {
283 |             chScan = 0;
284 |         }
285 |
286 |         return (int)((chScan << 8) + (chChar));
287 |     }
288 |     else
289 |     {
290 |         KBDKEYINFO ki;
291 |
292 |         ki.chChar = 0;
293 |         ki.chScan = 0;
294 |         KbdCharIn(&ki, IO_WAIT, 0);
295 |
296 |         if (ki.chChar == 0xe0)
297 |         {
298 |             if (ki.chScan)
299 |             {
300 |                 ki.chChar = 0;  /* force scan return */
```

```
301 |             }
302 |             else
303 |             {                   /* get next block */
304 |                 ki.chChar = 0;
305 |                 KbdCharIn(&ki, IO_WAIT, 0);
306 |                 if (!ki.chScan)
307 |                 {                   /* still no scan? */
308 |                     ki.chScan = ki.chChar;  /* move new char over */
309 |                     ki.chChar = 0;  /* force its return */
310 |                 }
311 |                 else
312 |                 {
313 |                     ki.chChar = 0;  /* force new scan */
314 |                 }
315 |             }
316 |         }
317 |         if (ki.chScan == 0xe0)
318 |         {
319 |             if (!ki.chChar)
320 |             {
321 |                 ki.chScan = 0;
322 |                 KbdCharIn(&ki, IO_WAIT, 0);
323 |                 if (!ki.chScan)
324 |                 {                   /* still no scan? */
325 |                     ki.chScan = ki.chChar;  /* move new char over */
326 |                     ki.chChar = 0;  /* force its return */
327 |                 }
328 |                 else
329 |                 {
330 |                     ki.chChar = 0;  /* force new scan */
331 |                 }
332 |             }
333 |             else
334 |             {
335 |                 ki.chScan = 0;  /* handle 0xe00d case */
336 |             }
337 |         }
338 |         if (ki.chChar)
339 |         {
340 |             ki.chScan = 0;
341 |         }
342 |
343 |         return (int)((ki.chScan << 8) + (ki.chChar));
344 |     }
345 | }
346 |
347 | char vm_getchxy(char x, char y)
348 | {
349 |     if (_osmode == DOS_MODE)
350 |     {
351 |         char cell[2];
352 |         vi_init();
353 |         v_getline(cell, (int)(x - 1), (int)(y - 1), 1);
354 |         return *cell;
355 |     }
356 |     else
357 |     {
358 |         char cell[2];
359 |         USHORT len = sizeof cell;
360 |         VioReadCharStr(cell, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
361 |         return *cell;
362 |     }
363 | }
364 |
365 | char vm_getattrxy(char x, char y)
366 | {
367 |     if (_osmode == DOS_MODE)
368 |     {
369 |         char cell[2];
370 |         vi_init();
371 |         v_getline(cell, (int)(x - 1), (int)(y - 1), 1);
372 |         return *(cell + 1);
373 |     }
374 |     else
375 |     {
376 |         char cell[2];
377 |         USHORT len = sizeof cell;
378 |         VioReadCellStr(cell, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
379 |         return *(cell + 1);
380 |     }
381 | }
382 |
383 | void vm_xgetchxy(char x, char y, char *attr, char *ch)
384 | {
385 |     if (_osmode == DOS_MODE)
386 |     {
387 |         char cell[2];
388 |         vi_init();
389 |         v_getline(cell, (int)(x - 1), (int)(y - 1), 1);
390 |         *ch = *cell;
391 |         *attr = *(cell + 1);
392 |     }
393 |     else
394 |     {
395 |         char cell[2];
396 |         USHORT len = sizeof cell;
397 |         VioReadCellStr(cell, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
398 |         *ch = *cell;
399 |         *attr = *(cell + 1);
400 |     }
```

```
401 | }
402 |
403 | void vm_setcursorstyle(int style)
404 | {
405 |     if (_osmode == DOS_MODE)
406 |     {
407 |         if (vm_iscolorcard())
408 |         {
409 |             switch (style)
410 |             {
411 |             case CURSORHALF:
412 |                 vm_setcursorsize(4, 7);
413 |                 break;
414 |             case CURSORFULL:
415 |                 vm_setcursorsize(0, 7);
416 |                 break;
417 |             case CURSORNORM:
418 |                 vm_setcursorsize(7, 8);
419 |                 break;
420 |             case CURSORHIDE:
421 |                 v_hidecursor();
422 |                 break;
423 |             default:
424 |                 break;
425 |             }
426 |         }
427 |         else
428 |         {
429 |             switch (style)
430 |             {
431 |             case CURSORHALF:
432 |                 vm_setcursorsize(8, 13);
433 |                 break;
434 |             case CURSORFULL:
435 |                 vm_setcursorsize(0, 13);
436 |                 break;
437 |             case CURSORNORM:
438 |                 vm_setcursorsize(11, 13);
439 |                 break;
440 |             case CURSORHIDE:
441 |                 vm_setcursorsize(32, 32);
442 |                 break;
443 |             default:
444 |                 break;
445 |             }
446 |         }
447 |     }
448 |     else
449 |     {
450 |         VIOCURSORINFO vi;
451 |
452 |         switch (style)
453 |         {
454 |         case CURSORHALF:
455 |             vi.yStart = -50;
456 |             vi.cEnd = -100;
457 |             vi.cx = 0;
458 |             vi.attr = 0;
459 |             VioSetCurType(&vi, 0);
460 |             break;
461 |         case CURSORFULL:
462 |             vi.yStart = 0;
463 |             vi.cEnd = -100;
464 |             vi.cx = 0;
465 |             vi.attr = 0;
466 |             VioSetCurType(&vi, 0);
467 |             break;
468 |         case CURSORNORM:
469 |             vi.yStart = -90;
470 |             vi.cEnd = -100;
471 |             vi.cx = 0;
472 |             vi.attr = 0;
473 |             VioSetCurType(&vi, 0);
474 |             break;
475 |         case CURSORHIDE:
476 |             vi.yStart = -90;
477 |             vi.cEnd = -100;
478 |             vi.cx = 0;
479 |             vi.attr = -1;
480 |             VioSetCurType(&vi, 0);
481 |             break;
482 |         default:
483 |             break;
484 |         }
485 |     }
486 | }
487 |
488 | void vm_putch(char x, char y, char ch)
489 | {
490 |     if (_osmode == DOS_MODE)
491 |     {
492 |         char cell[2];
493 |         vi_init();
494 |         v_getline(cell, (int)(x - 1), (int)(y - 1), 1);
495 |         *cell = ch;
496 |         v_putline(cell, (int)(x - 1), (int)(y - 1), 1);
497 |     }
498 |     else
499 |     {
500 |         VioWrtCharStr(&ch, 1, (USHORT) (y - 1), (USHORT) (x - 1), 0);
```

```
501 |     }
502 | }
503 |
504 | void vm_puts(char x, char y, char *str)
505 | {
506 |     if (_osmode == DOS_MODE)
507 |     {
508 |         vi_init();
509 |         while (*str)
510 |         {
511 |             vm_putch(x, y, *str);
512 |             str++;
513 |             x++;
514 |         }
515 |     }
516 |     else
517 |     {
518 |         VioWrtCharStr(str, (USHORT) strlen(str), (USHORT) (y - 1), (USHORT) (x - 1), 0);
519 |     }
520 | }
521 |
522 | void vm_xputch(char x, char y, char attr, char ch)
523 | {
524 |     if (_osmode == DOS_MODE)
525 |     {
526 |         char cell[2];
527 |         vi_init();
528 |         *cell = ch;
529 |         *(cell + 1) = attr;
530 |         v_putline(cell, (int)(x - 1), (int)(y - 1), 1);
531 |     }
532 |     else
533 |     {
534 |         VioWrtCharStrAtt(&ch, 1, (USHORT) (y - 1), (USHORT) (x - 1), (PBYTE) &attr, 0);
535 |     }
536 | }
537 |
538 | void vm_xputs(char x, char y, char attr, char *str)
539 | {
540 |     if (_osmode == DOS_MODE)
541 |     {
542 |         char *p, *cell, *pcell;
543 |         cell = malloc(strlen(str) * 2);
544 |         if (cell)
545 |         {
546 |             pcell = cell;
547 |             p = str;
548 |             while (*p)
549 |             {
550 |                 *pcell++ = *p++;
551 |                 *pcell++ = attr;
552 |             }
553 |             vi_init();
554 |             v_putline(cell, (int)(x - 1), (int)(y - 1), strlen(str));
555 |             free(cell);
556 |         }
557 |     }
558 |     else
559 |     {
560 |         VioWrtCharStrAtt(str, (USHORT) strlen(str), (USHORT) (y - 1), (USHORT) (x - 1), (PBYTE) &attr,
    |          0);
561 |     }
562 | }
563 |
564 | void vm_putattr(char x, char y, char attr)
565 | {
566 |     if (_osmode == DOS_MODE)
567 |     {
568 |         char cell[2];
569 |         vi_init();
570 |         v_getline(cell, (int)(x - 1), (int)(y - 1), 1);
571 |         *(cell + 1) = attr;
572 |         v_putline(cell, (int)(x - 1), (int)(y - 1), 1);
573 |     }
574 |     else
575 |     {
576 |         VioWrtNAttr((PBYTE) &attr, 1, (USHORT) (y - 1), (USHORT) (x - 1), 0);
577 |     }
578 | }
579 |
580 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr)
581 | {
582 |     if (_osmode == DOS_MODE)
583 |     {
584 |         char x, y;
585 |         for (y = y1; y <= y2; y++)
586 |         {
587 |             for (x = x1; x <= x2; x++)
588 |             {
589 |                 vm_xputch(x, y, attr, ' ');
590 |             }
591 |         }
592 |     }
593 |     else
594 |     {
595 |         char y, cell[2];
596 |         cell[0] = ' ';
597 |         cell[1] = attr;
598 |         for (y = y1; y <= y2; y++)
599 |         {
```

```
600 |             VioWrtNCell((PBYTE) &cell, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0)
    |             ;
601 |         }
602 |     }
603 | }
604 |
605 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr)
606 | {
607 |     if (_osmode == DOS_MODE)
608 |     {
609 |         char x, y;
610 |         for (y = y1; y <= y2; y++)
611 |         {
612 |             for (x = x1; x <= x2; x++)
613 |             {
614 |                 vm_putattr(x, y, attr);
615 |             }
616 |         }
617 |     }
618 |     else
619 |     {
620 |         char y;
621 |         for (y = y1; y <= y2; y++)
622 |         {
623 |             VioWrtNAttr((PBYTE) &attr, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0)
    |             ;
624 |         }
625 |     }
626 | }
627 |
628 | void vm_clearbox(char x1, char y1, char x2, char y2)
629 | {
630 |     if (_osmode == DOS_MODE)
631 |     {
632 |         char x, y;
633 |         for (y = y1; y <= y2; y++)
634 |         {
635 |             for (x = x1; x <= x2; x++)
636 |             {
637 |                 vm_putch(x, y, ' ');
638 |             }
639 |         }
640 |     }
641 |     else
642 |     {
643 |         char y, ch;
644 |         ch = ' ';
645 |         for (y = y1; y <= y2; y++)
646 |         {
647 |             VioWrtNChar((PBYTE) &ch, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
648 |         }
649 |     }
650 | }
651 |
652 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch)
653 | {
654 |     if (_osmode == DOS_MODE)
655 |     {
656 |         char x, y;
657 |         for (y = y1; y <= y2; y++)
658 |         {
659 |             for (x = x1; x <= x2; x++)
660 |             {
661 |                 vm_putch(x, y, ch);
662 |             }
663 |         }
664 |     }
665 |     else
666 |     {
667 |         char y;
668 |         for (y = y1; y <= y2; y++)
669 |         {
670 |             VioWrtNChar((PBYTE) &ch, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
671 |         }
672 |     }
673 | }
674 |
675 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest)
676 | {
677 |     if (_osmode == DOS_MODE)
678 |     {
679 |         char x, y;
680 |         for (y = y1; y <= y2; y++)
681 |         {
682 |             for (x = x1; x <= x2; x++)
683 |             {
684 |                 vm_xgetchxy(x, y, dest + 1, dest);
685 |                 dest += 2;
686 |             }
687 |         }
688 |     }
689 |     else
690 |     {
691 |         USHORT width;
692 |         char y;
693 |         width = (USHORT) ((x2 - x1 + 1) * 2);
694 |         for (y = y1; y <= y2; y++)
695 |         {
696 |             VioReadCellStr((PBYTE) dest, &width, (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
697 |             dest += width;
```

```
698 |         }
699 |     }
700 | }
701 |
702 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce)
703 | {
704 |     if (_osmode == DOS_MODE)
705 |     {
706 |         char x, y;
707 |         for (y = y1; y <= y2; y++)
708 |         {
709 |             for (x = x1; x <= x2; x++)
710 |             {
711 |                 vm_xputch(x, y, *(srce + 1), *srce);
712 |                 srce += 2;
713 |             }
714 |         }
715 |     }
716 |     else
717 |     {
718 |         USHORT width;
719 |         char y;
720 |         width = (USHORT) ((x2 - x1 + 1) * 2);
721 |         for (y = y1; y <= y2; y++)
722 |         {
723 |             VioWrtCellStr((PBYTE) srce, width, (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
724 |             srce += width;
725 |         }
726 |     }
727 | }
728 |
729 | void vm_horizline(char x1, char x2, char row, char attr, char ch)
730 | {
731 |     if (_osmode == DOS_MODE)
732 |     {
733 |         char x;
734 |         for (x = x1; x <= x2; x++)
735 |         {
736 |             vm_xputch(x, row, attr, ch);
737 |         }
738 |     }
739 |     else
740 |     {
741 |         char cell[2];
742 |         cell[0] = ch;
743 |         cell[1] = attr;
744 |         VioWrtNCell((PBYTE) &cell, (USHORT) (x2 - x1 + 1), (USHORT) (row - 1), (USHORT) (x1 - 1), 0);
745 |     }
746 | }
```

TEXT STATISTICS

17005 characters
746 lines

LEXICAL STATISTICS

25 comments [std-C]
0 comments [C++]
12 preprocessor instructions
4 constants [character]
2 constants [string]
204 constants [numeric]

SYMBOL TABLE

```
    CURSORFULL              414     434     461
    CURSORHALF              411     431     454
    CURSORHIDE              420     440     475
    CURSORNORM              417     437     468
    DOS_MODE         39      63      81     104     121     144     157     175     203     227     349
           367     385     405     490     506     524     540     566     582     607     630     654
           677     704     731
    INCL_DOSPROCESS          17
    INCL_KBD         15
    INCL_VIO         16
    IO_WAIT         294     305     322
    KBDKEYINFO              218     290
    KBDTRF_FINAL_CHAR_IN            221
    KbdCharIn       294     305     322
    KbdPeek         220
    PBYTE           534     560     576     600     623     647     670     696     723     744
    REGS            194     205
    USHORT          113     130     151+    359     360+    377     378+    396     397+    500+    518+
           534+    560+    576+    600+    623+    647+    670+    691     693     696+    718     720
           723+    744+
    VIOCURSORINFO           164     185     450
    VIOMODEINFO              72      73      90      91
    V_COLOR_12              139
    V_COLOR_8       139
    VioGetCurPos            114     131
    VioGetCurType           186
    VioGetMode               74      92
    VioReadCellStr          378     397     696
    VioReadCharStr          360
    VioSetCurPos            151
    VioSetCurType           169     459     466     473     480
    VioWrtCellStr           723
    VioWrtCharStr           500     518
    VioWrtCharStrAtt        534     560
    VioWrtNAttr             576     623
    VioWrtNCell             600     744
    VioWrtNChar             647     670
    _int86          196     213
    _osmode          39      63      81     104     121     144     157     175     203     227     349
           367     385     405     490     506     524     540     566     582     607     630     654
           677     704     731
    ah              195     197     212
    al              198
    attr             38      55     168     383     391     399     458     465     472     479     522
           529     534     538     551     560     564     571     576     580     589     597     605
           614     623     729     736     743
    cEnd            166     188     456     463     470     477
    cb               73      91
    cell            351     353     354     358     359     360     361     369     371     372     376
           377     378     379     387     389     390     391     395     396     397     398     399
           492     494     495     496     526     528     529     530     542     543     544     546
           554     555     568     570     571     572     595     596     597     600     741     742
           743     744
    cend            177     179     181
    ch              383     390     398     488     495     500     522     528     534     643     644
           647     652     661     670     729     736     742
    chChar           27     192     198     229     236     237     241     245     247     250     251
           255     261     265     268     269     273     281     286     292     296     300     304
           308     309     313     319     325     326     330     338     343
    chScan           27     192     197     229     236     239     247     248     250     259     263
           265     266     268     278     283     286     293     298     306     308     317     321
           323     325     335     340     343
    col              75     106     108     109     113     114     115     123     125     130     131
    counter         206     207     211
    cstart          177     179     180
    cur_end          47      58
    cur_start        47      58
    cx              167     457     464     471     478
    dest            675     684+    685     696     697
    dos              12
    end              25      26     155     160     166     173     181     188
    fbStatus        219     221
    flags           214
    free            555
    h       10       11      12      13      19     195     197     198     212
    height           56      65      67      83      85      86
    hw              138     139+
    ki              218     219     220     221     290     292     293     294     296     298     300
           304     305     306     308+    309     313     317     319     321     322     323     325+
           326     330     335     338     340     343+
    len             359     360     377     378     396     397
    malloc          543
    opsysDetect              41
    opsysTimeSlice          209
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VMGROS2.C; VidMgr module for OS/2 compilers.  Release 1.2.
 *
 *  This module written in March 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in May 1996.
 */

#define INCL_KBD
#define INCL_VIO
#define INCL_DOSPROCESS

#include <string.h>
#include <os2.h>
#include "vidmgr.h"

#ifndef KBDTRF_FINAL_CHAR_IN
#define KBDTRF_FINAL_CHAR_IN  FINAL_CHAR_IN
#endif

static void vm_setcursorsize(char start, char end);
static void vm_getcursorsize(char *start, char *end);

void vm_init(void)
{
    vm_getinfo(&vm_startup);
    vm_setattr(vm_startup.attr);
}

void vm_done(void)
{
    vm_setcursorsize(vm_startup.cur_start, vm_startup.cur_end);
}

void vm_getinfo(struct vm_info *v)
{
    v->ypos = vm_wherey();
    v->xpos = vm_wherex();
    v->attr = vm_getattrxy(v->xpos, v->ypos);
    v->height = vm_getscreenheight();
    v->width = vm_getscreenwidth();
    vm_getcursorsize(&v->cur_start, &v->cur_end);
}

char vm_getscreenwidth(void)
{
    VIOMODEINFO vi;
    vi.cb = sizeof(VIOMODEINFO);
    VioGetMode(&vi, 0);
    return vi.col;
}

char vm_getscreenheight(void)
{
    VIOMODEINFO vi;
    vi.cb = sizeof(VIOMODEINFO);
    VioGetMode(&vi, 0);
    return vi.row;
}

short vm_getscreensize(void)
{
    return (short)(vm_getscreenwidth() * vm_getscreenheight() * 2);
}

char vm_wherex(void)
{
    USHORT row, col;
    VioGetCurPos(&row, &col, 0);
    return (char)(col + 1);
}

char vm_wherey(void)
{
    USHORT row, col;
    VioGetCurPos(&row, &col, 0);
    return (char)(row + 1);
}

void vm_gotoxy(char x, char y)
{
    VioSetCurPos((USHORT) (y - 1), (USHORT) (x - 1), 0);
}

static void vm_setcursorsize(char start, char end)
{
    VIOCURSORINFO vi;
    vi.yStart = start;
    vi.cEnd = end;
    vi.cx = 0;
    vi.attr = 0;
    VioSetCurType(&vi, 0);
}

static void vm_getcursorsize(char *start, char *end)
{
    VIOCURSORINFO vi;
    VioGetCurType(&vi, 0);
    *start = vi.yStart;
```

```
101 |     *end = vi.cEnd;
102 | }
103 |
104 | int vm_kbhit(void)
105 | {
106 |     KBDKEYINFO ki;
107 |     ki.fbStatus = 0;
108 |     KbdPeek(&ki, 0);
109 |     return ki.fbStatus & KBDTRF_FINAL_CHAR_IN;
110 | }
111 |
112 | int vm_getch(void)
113 | {
114 |     KBDKEYINFO ki;
115 |
116 |     ki.chChar = 0;
117 |     ki.chScan = 0;
118 |     KbdCharIn(&ki, IO_WAIT, 0);
119 |
120 |     if (ki.chChar == 0xe0)
121 |     {
122 |         if (ki.chScan)
123 |         {
124 |             ki.chChar = 0;      /* force scan return */
125 |         }
126 |         else
127 |         {                       /* get next block */
128 |             ki.chChar = 0;
129 |             KbdCharIn(&ki, IO_WAIT, 0);
130 |             if (!ki.chScan)
131 |             {                   /* still no scan? */
132 |                 ki.chScan = ki.chChar;  /* move new char over */
133 |                 ki.chChar = 0;  /* force its return */
134 |             }
135 |             else
136 |             {
137 |                 ki.chChar = 0;  /* force new scan */
138 |             }
139 |         }
140 |     }
141 |     if (ki.chScan == 0xe0)
142 |     {
143 |         if (!ki.chChar)
144 |         {
145 |             ki.chScan = 0;
146 |             KbdCharIn(&ki, IO_WAIT, 0);
147 |             if (!ki.chScan)
148 |             {                   /* still no scan? */
149 |                 ki.chScan = ki.chChar;  /* move new char over */
150 |                 ki.chChar = 0;  /* force its return */
151 |             }
152 |             else
153 |             {
154 |                 ki.chChar = 0;  /* force new scan */
155 |             }
156 |         }
157 |         else
158 |         {
159 |             ki.chScan = 0;      /* handle 0xe00d case */
160 |         }
161 |     }
162 |     if (ki.chChar)
163 |     {
164 |         ki.chScan = 0;
165 |     }
166 |
167 |     return (int)((ki.chScan << 8) + (ki.chChar));
168 | }
169 |
170 | char vm_getchxy(char x, char y)
171 | {
172 |     char ch;
173 |     USHORT len = 1;
174 |     VioReadCharStr(&ch, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
175 |     return ch;
176 | }
177 |
178 | char vm_getattrxy(char x, char y)
179 | {
180 |     char cell[4];
181 |     USHORT len = 4;
182 |     VioReadCellStr(cell, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
183 |     return *(cell + 1);
184 | }
185 |
186 | void vm_xgetchxy(char x, char y, char *attr, char *ch)
187 | {
188 |     char cell[4];
189 |     USHORT len = 4;
190 |     VioReadCellStr(cell, &len, (USHORT) (y - 1), (USHORT) (x - 1), 0);
191 |     *ch = *cell;
192 |     *attr = *(cell + 1);
193 | }
194 |
195 | void vm_setcursorstyle(int style)
196 | {
197 |     VIOCURSORINFO vi;
198 |
199 |     switch (style)
200 |     {
```

```
201 |     case CURSORHALF:
202 |         vi.yStart = -50;
203 |         vi.cEnd = -100;
204 |         vi.cx = 0;
205 |         vi.attr = 0;
206 |         VioSetCurType(&vi, 0);
207 |         break;
208 |     case CURSORFULL:
209 |         vi.yStart = 0;
210 |         vi.cEnd = -100;
211 |         vi.cx = 0;
212 |         vi.attr = 0;
213 |         VioSetCurType(&vi, 0);
214 |         break;
215 |     case CURSORNORM:
216 |         vi.yStart = -90;
217 |         vi.cEnd = -100;
218 |         vi.cx = 0;
219 |         vi.attr = 0;
220 |         VioSetCurType(&vi, 0);
221 |         break;
222 |     case CURSORHIDE:
223 |         vi.yStart = -90;
224 |         vi.cEnd = -100;
225 |         vi.cx = 0;
226 |         vi.attr = -1;
227 |         VioSetCurType(&vi, 0);
228 |         break;
229 |     default:
230 |         break;
231 |     }
232 | }
233 |
234 | void vm_putch(char x, char y, char ch)
235 | {
236 |     VioWrtCharStr(&ch, 1, (USHORT) (y - 1), (USHORT) (x - 1), 0);
237 | }
238 |
239 | void vm_puts(char x, char y, char *str)
240 | {
241 |     VioWrtCharStr(str, (USHORT) strlen(str), (USHORT) (y - 1), (USHORT) (x - 1), 0);
242 | }
243 |
244 | void vm_xputch(char x, char y, char attr, char ch)
245 | {
246 |     VioWrtCharStrAtt(&ch, 1, (USHORT) (y - 1), (USHORT) (x - 1), (PBYTE) &attr, 0);
247 | }
248 |
249 | void vm_xputs(char x, char y, char attr, char *str)
250 | {
251 |     VioWrtCharStrAtt(str, (USHORT) strlen(str), (USHORT) (y - 1), (USHORT) (x - 1), (PBYTE) &attr, 0);
252 | }
253 |
254 | void vm_putattr(char x, char y, char attr)
255 | {
256 |     VioWrtNAttr((PBYTE) &attr, 1, (USHORT) (y - 1), (USHORT) (x - 1), 0);
257 | }
258 |
259 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr)
260 | {
261 |     char y, cell[2];
262 |     cell[0] = ' ';
263 |     cell[1] = attr;
264 |     for (y = y1; y <= y2; y++)
265 |     {
266 |         VioWrtNCell((PBYTE) &cell, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
267 |     }
268 | }
269 |
270 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr)
271 | {
272 |     char y;
273 |     for (y = y1; y <= y2; y++)
274 |     {
275 |         VioWrtNAttr((PBYTE) &attr, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
276 |     }
277 | }
278 |
279 | void vm_clearbox(char x1, char y1, char x2, char y2)
280 | {
281 |     char y, ch;
282 |     ch = ' ';
283 |     for (y = y1; y <= y2; y++)
284 |     {
285 |         VioWrtNChar((PBYTE) &ch, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
286 |     }
287 | }
288 |
289 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch)
290 | {
291 |     char y;
292 |     for (y = y1; y <= y2; y++)
293 |     {
294 |         VioWrtNChar((PBYTE) &ch, (USHORT) (x2 - x1 + 1), (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
295 |     }
296 | }
297 |
298 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest)
299 | {
300 |     USHORT width;
```

```
301 |     char y;
302 |     width = (USHORT) ((x2 - x1 + 1) * 2);
303 |     for (y = y1; y <= y2; y++)
304 |     {
305 |         VioReadCellStr((PBYTE) dest, &width, (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
306 |         dest += width;
307 |     }
308 | }
309 |
310 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce)
311 | {
312 |     USHORT width;
313 |     char y;
314 |     width = (USHORT) ((x2 - x1 + 1) * 2);
315 |     for (y = y1; y <= y2; y++)
316 |     {
317 |         VioWrtCellStr((PBYTE) srce, width, (USHORT) (y - 1), (USHORT) (x1 - 1), 0);
318 |         srce += width;
319 |     }
320 | }
321 |
322 | void vm_horizline(char x1, char x2, char row, char attr, char ch)
323 | {
324 |     char cell[2];
325 |     cell[0] = ch;
326 |     cell[1] = attr;
327 |     VioWrtNCell((PBYTE) &cell, (USHORT) (x2 - x1 + 1), (USHORT) (row - 1), (USHORT) (x1 - 1), 0);
328 | }
```

```
TEXT STATISTICS

  7511 characters
   328 lines

LEXICAL STATISTICS

    13 comments [std-C]
     0 comments [C++]
     9 preprocessor instructions
     2 constants [character]
     1 constants [string]
   123 constants [numeric]

SYMBOL TABLE

    CURSORFULL             208
    CURSORHALF             201
    CURSORHIDE             222
    CURSORNORM             215
    FINAL_CHAR_IN           19
    INCL_DOSPROCESS         12
    INCL_KBD        10
    INCL_VIO        11
    IO_WAIT        118     129     146
    KBDKEYINFO             106     114
    KBDTRF_FINAL_CHAR_IN            18      19     109
    KbdCharIn      118     129     146
    KbdPeek        108
    PBYTE          246     251     256     266     275     285     294     305     317     327
    USHORT          69      76      83+    173     174+    181     182+    189     190+    236+    241+
           246+    251+    256+    266+    275+    285+    294+    300     302     305+    312     314
           317+    327+
    VIOCURSORINFO           88      98     197
    VIOMODEINFO             48      49      56      57
    VioGetCurPos            70      77
    VioGetCurType           99
    VioGetMode              50      58
    VioReadCellStr         182     190     305
    VioReadCharStr         174
    VioSetCurPos            83
    VioSetCurType           93     206     213     220     227
    VioWrtCellStr          317
    VioWrtCharStr          236     241
    VioWrtCharStrAtt       246     251
    VioWrtNAttr            256     275
    VioWrtNCell            266     327
    VioWrtNChar            285     294
    attr            28      40      92     186     192     205     212     219     226     244     246
           249     251     254     256     259     263     270     275     322     326
    cEnd            90     101     203     210     217     224
    cb              49      57
    cell           180     182     183     188     190     191     192     261     262     263     266
           324     325     326     327
    ch             172     174     175     186     191     234     236     244     246     281     282
           285     289     294     322     325
    chChar         116     120     124     128     132     133     137     143     149     150     154
           162     167
    chScan         117     122     130     132     141     145     147     149     159     164     167
    col             51      69      70      71      76      77
    cur_end         33      43
    cur_start       33      43
    cx              91     204     211     218     225
    dest           298     305     306
    end             22      23      86      90      96     101
    fbStatus       107     109
    h       14      15
    height          41
    ki             106     107     108     109     114     116     117     118     120     122     124
           128     129     130     132+    133     137     141     143     145     146     147     149+
           150     154     159     162     164     167+
    len            173     174     181     182     189     190
    os2             15
    row             59      69      70      76      77      78     322     327
    srce           310     317     318
    start           22      23      86      89      96     100
    str            239     241+    249     251+
    string          14
    strlen         241     251
    style          195     199
    v       36      38      39      40+     41      42      43+
    vi              48      49      50      51      56      57      58      59      88      89      90
            91      92      93      98      99     100     101     197     202     203     204     205
           206     209     210     211     212     213     216     217     218     219     220     223
           224     225     226     227
    vm_clearbox            279
    vm_done         31
    vm_fillbox             289
    vm_getattrxy            40     178
    vm_getch       112
    vm_getchxy             170
    vm_getcursorsize        23      43      96
    vm_getinfo              27      36
    vm_getscreenheight              41      54      64
    vm_getscreensize        62
    vm_getscreenwidth       42      46      64
    vm_gettext             298
    vm_gotoxy       81
    vm_horizline           322
    vm_info         36
```

```
vm_init          25
vm_kbhit        104
vm_paintbox             270
vm_paintclearbox        259
vm_putattr              254
vm_putch        234
vm_puts         239
vm_puttext              310
vm_setattr               28
vm_setcursorsize         22       33       86
vm_setcursorstyle       195
vm_startup               27       28       33+
vm_wherex        39      67
vm_wherey        38      74
vm_xgetchxy             186
vm_xputch       244
vm_xputs        249
width            42      300      302      305      306      312      314      317      318
x        81      83      170      174      178      182      186      190      234      236      239      241
         244     246     249      251      254      256
x1               259     266+     270      275+     279      285+     289      294+     298      302      305
         310     314     317      322      327+
x2               259     266      270      275      279      285      289      294      298      302      310
         314     322     327
xpos             39      40
y        81      83      170      174      178      182      186      190      234      236      239      241
         244     246     249      251      254      256      261      264+     266      272      273+     275
         281     283+    285      291      292+     294      301      303+     305      313      315+     317
y1               259     264      270      273      279      283      289      292      298      303      310
         315
y2               259     264      270      273      279      283      289      292      298      303      310
         315
yStart           89      100      202      209      216      223
ypos             38      40
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  VMGRWNT.C; VidMgr module for Windows 95/NT compilers.  Release 1.3.
 *
 *  This module written in May 1996 by Andrew Clarke and released to the
 *  public domain.  Last modified in October 1996.
 */

#include <stdlib.h>
#include <string.h>

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include "vidmgr.h"

#ifdef __CYGWIN32__
typedef WORD far *LPWORD;
#endif

static HANDLE HInput = INVALID_HANDLE_VALUE;
static HANDLE HOutput = INVALID_HANDLE_VALUE;
static unsigned long key_hit = 0xFFFFFFFFUL;

static int video_init = 0;

#define vi_init()  if (!video_init) vm_vi_init()
#define vi_done()  if (video_init) vm_vi_done()

void vm_vi_init(void)
{
    HInput = GetStdHandle(STD_INPUT_HANDLE);
    HOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    video_init = 1;
}

void vm_vi_done(void)
{
    CloseHandle(HInput);
    HInput = INVALID_HANDLE_VALUE;
    CloseHandle(HOutput);
    HOutput = INVALID_HANDLE_VALUE;
    video_init = 0;
}

void vm_init(void)
{
    vi_init();
    vm_getinfo(&vm_startup);
    vm_setattr(vm_startup.attr);
}

void vm_done(void)
{
    CONSOLE_CURSOR_INFO cci;
    SetConsoleCursorInfo(HOutput, &cci);
    vm_startup.dwSize = (unsigned short)cci.dwSize;
    vm_startup.bVisible = (int)cci.bVisible;
    vi_done();
}

void vm_getinfo(struct vm_info *v)
{
    CONSOLE_CURSOR_INFO cci;
    v->ypos = vm_wherey();
    v->xpos = vm_wherex();
    v->attr = vm_getattrxy(v->xpos, v->ypos);
    v->height = vm_getscreenheight();
    v->width = vm_getscreenwidth();
    cci.dwSize = (DWORD) vm_startup.dwSize;
    cci.bVisible = (BOOL) vm_startup.bVisible;
    GetConsoleCursorInfo(HOutput, &cci);
}

char vm_getscreenwidth(void)
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(HOutput, &csbi);
    return (char)csbi.dwSize.X;
}

char vm_getscreenheight(void)
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(HOutput, &csbi);
    return (char)csbi.dwSize.Y;
}

short vm_getscreensize(void)
{
    return (short)(vm_getscreenwidth() * vm_getscreenheight() * 2);
}

char vm_wherex(void)
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(HOutput, &csbi);
    return (char)(csbi.dwCursorPosition.X + 1);
}
```

```
101 |
102 | char vm_wherey(void)
103 | {
104 |     CONSOLE_SCREEN_BUFFER_INFO csbi;
105 |     GetConsoleScreenBufferInfo(HOutput, &csbi);
106 |     return (char)(csbi.dwCursorPosition.Y + 1);
107 | }
108 |
109 | void vm_gotoxy(char x, char y)
110 | {
111 |     COORD dwCursorPosition;
112 |     dwCursorPosition.X = (SHORT) (x - 1);
113 |     dwCursorPosition.Y = (SHORT) (y - 1);
114 |     SetConsoleCursorPosition(HOutput, dwCursorPosition);
115 | }
116 |
117 | int vm_kbhit(void)
118 | {
119 |     int iKey = 0;
120 |     INPUT_RECORD irBuffer;
121 |     DWORD pcRead;
122 |
123 |     if (key_hit != 0xFFFFFFFFUL)
124 |     {
125 |         return (int)key_hit;
126 |     }
127 |
128 |     memset(&irBuffer, 0, sizeof irBuffer);
129 |
130 |     if (WaitForSingleObject(HInput, 0L) == 0)
131 |     {
132 |         ReadConsoleInput(HInput, &irBuffer, 1, &pcRead);
133 |         if (irBuffer.EventType == KEY_EVENT &&
134 |           irBuffer.Event.KeyEvent.bKeyDown != 0 &&
135 |           irBuffer.Event.KeyEvent.wRepeatCount <= 1)
136 |         {
137 |             WORD vk, vs, uc;
138 |             BOOL fShift, fAlt, fCtrl;
139 |
140 |             vk = irBuffer.Event.KeyEvent.wVirtualKeyCode;
141 |             vs = irBuffer.Event.KeyEvent.wVirtualScanCode;
142 | #ifdef __CYGWIN32__
143 |             uc = irBuffer.Event.KeyEvent.AsciiChar;
144 | #else
145 |             uc = irBuffer.Event.KeyEvent.uChar.AsciiChar;
146 | #endif
147 |
148 |             fShift = (irBuffer.Event.KeyEvent.dwControlKeyState & SHIFT_PRESSED);
149 |             fAlt = (irBuffer.Event.KeyEvent.dwControlKeyState & (RIGHT_ALT_PRESSED + LEFT_ALT_PRESSED)
    |             );
150 |             fCtrl = (irBuffer.Event.KeyEvent.dwControlKeyState & (RIGHT_CTRL_PRESSED + LEFT_CTRL_PRESS
    |             ED));
151 |
152 |             if (uc == 0)
153 |             {                    /* function keys */
154 |                 switch (vk)
155 |                 {
156 |                 case 0x21:     /* PgUp */
157 |                     if (fCtrl)
158 |                     {
159 |                         vs = 0x84;  /* Ctrl+PgUp */
160 |                     }
161 |                     break;
162 |
163 |                 case 0x22:     /* PgDn */
164 |                     if (fCtrl)
165 |                     {
166 |                         vs = 0x76;  /* Ctrl+PgDn */
167 |                     }
168 |                     break;
169 |
170 |                 case 0x23:     /* End */
171 |                     if (fCtrl)
172 |                     {
173 |                         vs = 0x75;  /* Ctrl+End */
174 |                     }
175 |                     break;
176 |
177 |                 case 0x24:     /* Home */
178 |                     if (fCtrl)
179 |                     {
180 |                         vs = 0x77;  /* Ctrl+Home */
181 |                     }
182 |                     break;
183 |
184 |                 case 0x25:     /* Left Arrow */
185 |                     if (fCtrl)
186 |                     {
187 |                         vs = 0x73;  /* Ctrl+Left Arrow */
188 |                     }
189 |                     break;
190 |
191 |                 case 0x26:     /* Up Arrow */
192 |                     if (fCtrl)
193 |                     {
194 |                         vs = 0x8d;  /* Ctrl+Up Arrow */
195 |                     }
196 |                     break;
197 |
198 |                 case 0x27:     /* Right Arrow */
```

```
199 |                     if (fCtrl)
200 |                     {
201 |                         vs = 0x74;  /* Ctrl+Right Arrow */
202 |                     }
203 |                     break;
204 |
205 |                 case 0x28:     /* Down Arrow */
206 |                     if (fCtrl)
207 |                     {
208 |                         vs = 0x91;  /* Ctrl+Down Arrow */
209 |                     }
210 |                     break;
211 |
212 |                 case 0x70:     /* F-Keys */
213 |                 case 0x71:
214 |                 case 0x72:
215 |                 case 0x73:
216 |                 case 0x74:
217 |                 case 0x75:
218 |                 case 0x76:
219 |                 case 0x77:
220 |                 case 0x78:
221 |                 case 0x79:
222 |                     if (fAlt)
223 |                     {
224 |                         vs += 0x2d;  /* Alt+F-Key */
225 |                     }
226 |                     else if (fShift)
227 |                     {
228 |                         vs += 0x19;  /* Shift+F-Key */
229 |                     }
230 |                     break;
231 |                 }
232 |
233 |                 if (vk > 0x20 && vk < 0x92)  /* If it's OK use scan code */
234 |                 {
235 |                     iKey = vs << 8;
236 |                 }
237 |             }
238 |             else
239 |             {
240 |                 if (fAlt)       /* Alt+Key */
241 |                 {
242 |                     iKey = vs << 8;
243 |                 }
244 |                 else if (fCtrl) /* Ctrl+Key */
245 |                 {
246 |                     iKey = vk & 0xbf;
247 |                 }
248 |                 else
249 |                 {
250 |                     iKey = uc;
251 |                 }
252 |             }
253 |         }
254 |     }
255 |
256 |     if (iKey != 0)
257 |     {
258 |         key_hit = iKey;
259 |     }
260 |
261 |     return (int)iKey;
262 | }
263 |
264 | int vm_getch(void)
265 | {
266 |     int iKey;
267 |     while (key_hit == 0xFFFFFFFFUL)
268 |     {
269 |         vm_kbhit();
270 |     }
271 |     iKey = key_hit;
272 |     key_hit = 0xFFFFFFFFUL;
273 |     return (int)iKey;
274 | }
275 |
276 | char vm_getchxy(char x, char y)
277 | {
278 |     char ch;
279 |     DWORD len;
280 |     COORD coord;
281 |     coord.X = (DWORD) (x - 1);
282 |     coord.Y = (DWORD) (y - 1);
283 |     ReadConsoleOutputCharacterA(HOutput, &ch, 1, coord, &len);
284 |     return ch;
285 | }
286 |
287 | char vm_getattrxy(char x, char y)
288 | {
289 |     DWORD len;
290 |     COORD coord;
291 |     WORD wattr;
292 |     coord.X = (DWORD) (x - 1);
293 |     coord.Y = (DWORD) (y - 1);
294 |     ReadConsoleOutputAttribute(HOutput, &wattr, 1, coord, &len);
295 |     return (char)wattr;
296 | }
297 |
298 | void vm_xgetchxy(char x, char y, char *attr, char *ch)
```

```
{
    DWORD len;
    COORD coord;
    WORD wattr;
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    ReadConsoleOutputCharacterA(HOutput, ch, 1, coord, &len);
    ReadConsoleOutputAttribute(HOutput, &wattr, 1, coord, &len);
    *attr = (char)wattr;
}

void vm_setcursorstyle(int style)
{
    CONSOLE_CURSOR_INFO cci;
    GetConsoleCursorInfo(HOutput, &cci);
    switch (style)
    {
    case CURSORHALF:
        cci.bVisible = 1;
        cci.dwSize = 49;
        SetConsoleCursorInfo(HOutput, &cci);
        break;
    case CURSORFULL:
        cci.bVisible = 1;
        cci.dwSize = 99;
        SetConsoleCursorInfo(HOutput, &cci);
        break;
    case CURSORNORM:
        cci.bVisible = 1;
        cci.dwSize = 12;
        SetConsoleCursorInfo(HOutput, &cci);
        break;
    case CURSORHIDE:
        cci.bVisible = 0;
        SetConsoleCursorInfo(HOutput, &cci);
        break;
    default:
        break;
    }
}

void vm_putch(char x, char y, char ch)
{
    DWORD len;
    COORD coord;
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    WriteConsoleOutputCharacterA(HOutput, &ch, 1, coord, &len);
}

void vm_puts(char x, char y, char *str)
{
    DWORD len;
    COORD coord;
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    WriteConsoleOutputCharacterA(HOutput, str, (DWORD) strlen(str), coord, &len);
}

void vm_xputch(char x, char y, char attr, char ch)
{
    DWORD len;
    COORD coord;
    WORD wattr;
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    wattr = attr;
    WriteConsoleOutputCharacterA(HOutput, &ch, 1, coord, &len);
    WriteConsoleOutputAttribute(HOutput, &wattr, 1, coord, &len);
}

void vm_xputs(char x, char y, char attr, char *str)
{
    DWORD i, len;
    COORD coord;
    LPWORD pwattr;
    pwattr = malloc(strlen(str) * sizeof(*pwattr));
    if (!pwattr)
    {
        return;
    }
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    for (i = 0; i < strlen(str); i++)
    {
        *(pwattr + i) = attr;
    }
    WriteConsoleOutputCharacterA(HOutput, str, (DWORD) strlen(str), coord, &len);
    WriteConsoleOutputAttribute(HOutput, pwattr, (DWORD) strlen(str), coord, &len);
    free(pwattr);
}

void vm_putattr(char x, char y, char attr)
{
    DWORD len;
    COORD coord;
    WORD wattr;
    coord.X = (DWORD) (x - 1);
    coord.Y = (DWORD) (y - 1);
    wattr = attr;
```

```
399 |     WriteConsoleOutputAttribute(HOutput, &wattr, 1, coord, &len);
400 | }
401 |
402 | void vm_paintclearbox(char x1, char y1, char x2, char y2, char attr)
403 | {
404 |     COORD coord;
405 |     LPWORD pwattr;
406 |     char y, *pstr;
407 |     DWORD i, len, width;
408 |     width = (x2 - x1 + 1);
409 |     pwattr = malloc(width * sizeof(*pwattr));
410 |     if (!pwattr)
411 |     {
412 |         return;
413 |     }
414 |     pstr = malloc(width);
415 |     if (!pstr)
416 |     {
417 |         free(pwattr);
418 |         return;
419 |     }
420 |     for (i = 0; i < width; i++)
421 |     {
422 |         *(pwattr + i) = attr;
423 |         *(pstr + i) = ' ';
424 |     }
425 |     for (y = y1; y <= y2; y++)
426 |     {
427 |         coord.X = (DWORD) (x1 - 1);
428 |         coord.Y = (DWORD) (y - 1);
429 |         WriteConsoleOutputCharacterA(HOutput, pstr, width, coord, &len);
430 |         WriteConsoleOutputAttribute(HOutput, pwattr, width, coord, &len);
431 |     }
432 |     free(pwattr);
433 |     free(pstr);
434 | }
435 |
436 | void vm_paintbox(char x1, char y1, char x2, char y2, char attr)
437 | {
438 |     DWORD i, len, width;
439 |     COORD coord;
440 |     LPWORD pwattr;
441 |     char y;
442 |     width = (x2 - x1 + 1);
443 |     pwattr = malloc(width * sizeof(*pwattr));
444 |     if (!pwattr)
445 |     {
446 |         return;
447 |     }
448 |     for (i = 0; i < width; i++)
449 |     {
450 |         *(pwattr + i) = attr;
451 |     }
452 |     for (y = y1; y <= y2; y++)
453 |     {
454 |         coord.X = (DWORD) (x1 - 1);
455 |         coord.Y = (DWORD) (y - 1);
456 |         WriteConsoleOutputAttribute(HOutput, pwattr, width, coord, &len);
457 |     }
458 |     free(pwattr);
459 | }
460 |
461 | void vm_clearbox(char x1, char y1, char x2, char y2)
462 | {
463 |     vm_fillbox(x1, y1, x2, y2, ' ');
464 | }
465 |
466 | void vm_fillbox(char x1, char y1, char x2, char y2, char ch)
467 | {
468 |     DWORD i, len, width;
469 |     COORD coord;
470 |     char y, *pstr;
471 |     width = (x2 - x1 + 1);
472 |     pstr = malloc(width);
473 |     if (!pstr)
474 |     {
475 |         return;
476 |     }
477 |     for (i = 0; i < width; i++)
478 |     {
479 |         *(pstr + i) = ch;
480 |     }
481 |     for (y = y1; y <= y2; y++)
482 |     {
483 |         coord.X = (DWORD) (x1 - 1);
484 |         coord.Y = (DWORD) (y - 1);
485 |         WriteConsoleOutputCharacterA(HOutput, pstr, width, coord, &len);
486 |     }
487 |     free(pstr);
488 | }
489 |
490 | void vm_gettext(char x1, char y1, char x2, char y2, char *dest)
491 | {
492 |     DWORD i, len, width;
493 |     COORD coord;
494 |     LPWORD pwattr;
495 |     char y, *pstr;
496 |     width = (x2 - x1 + 1);
497 |     pwattr = malloc(width * sizeof(*pwattr));
498 |     if (!pwattr)
```

```
499 |     {
500 |         return;
501 |     }
502 |     pstr = malloc(width);
503 |     if (!pstr)
504 |     {
505 |         free(pwattr);
506 |         return;
507 |     }
508 |     for (y = y1; y <= y2; y++)
509 |     {
510 |         coord.X = (DWORD) (x1 - 1);
511 |         coord.Y = (DWORD) (y - 1);
512 |         ReadConsoleOutputCharacterA(HOutput, pstr, width, coord, &len);
513 |         ReadConsoleOutputAttribute(HOutput, pwattr, width, coord, &len);
514 |         for (i = 0; i < width; i++)
515 |         {
516 |             *dest = *(pstr + i);
517 |             dest++;
518 |             *dest = (char)*(pwattr + i);
519 |             dest++;
520 |         }
521 |     }
522 |     free(pwattr);
523 |     free(pstr);
524 | }
525 |
526 | void vm_puttext(char x1, char y1, char x2, char y2, char *srce)
527 | {
528 |     DWORD i, len, width;
529 |     COORD coord;
530 |     LPWORD pwattr;
531 |     char y, *pstr;
532 |     width = (x2 - x1 + 1);
533 |     pwattr = malloc(width * sizeof(*pwattr));
534 |     if (!pwattr)
535 |     {
536 |         return;
537 |     }
538 |     pstr = malloc(width);
539 |     if (!pstr)
540 |     {
541 |         free(pwattr);
542 |         return;
543 |     }
544 |     for (y = y1; y <= y2; y++)
545 |     {
546 |         for (i = 0; i < width; i++)
547 |         {
548 |             *(pstr + i) = *srce;
549 |             srce++;
550 |             *(pwattr + i) = *srce;
551 |             srce++;
552 |         }
553 |         coord.X = (DWORD) (x1 - 1);
554 |         coord.Y = (DWORD) (y - 1);
555 |         WriteConsoleOutputCharacterA(HOutput, pstr, width, coord, &len);
556 |         WriteConsoleOutputAttribute(HOutput, pwattr, width, coord, &len);
557 |     }
558 |     free(pwattr);
559 |     free(pstr);
560 | }
561 |
562 | void vm_horizline(char x1, char x2, char row, char attr, char ch)
563 | {
564 |     char x;
565 |     for (x = x1; x <= x2; x++)
566 |     {
567 |         vm_xputch(x, row, attr, ch);
568 |     }
569 | }
```

TEXT STATISTICS

14067 characters
569 lines

LEXICAL STATISTICS

25 comments [std-C]
0 comments [C++]
12 preprocessor instructions
2 constants [character]
1 constants [string]
106 constants [numeric]

SYMBOL TABLE

```
    AsciiChar       143     145
    BOOL             72     138
    CONSOLE_CURSOR_INFO              56      65     312
    CONSOLE_SCREEN_BUFFER_INFO               78      85      97     104
    COORD           111     280     290     301     343     352     361     373     394     404     439
           469     493     529
    CURSORFULL              321
    CURSORHALF              316
    CURSORHIDE              331
    CURSORNORM              326
    CloseHandle              40      42
    DWORD            71     121     279     281     282     289     292     293     300     303     304
           342     344     345     351     353     354     355     360     363     364     372     380
           381     386     387     393     396     397     407     427     428     438     454     455
           468     483     484     492     510     511     528     553     554
    Event           134     135     140     141     143     145     148     149     150
    EventType       133
    GetConsoleCursorInfo             73     313
    GetConsoleScreenBufferInfo               79      86      98     105
    GetStdHandle             33      34
    HANDLE           22      23
    HInput           22      33      40      41     130     132
    HOutput          23      34      42      43      57      73      79      86      98     105     114
           283     294     305     306     313     319     324     329     333     346     355     366
           367     386     387     399     429     430     456     485     512     513     555     556
    INPUT_RECORD            120
    INVALID_HANDLE_VALUE             22      23      41      43
    KEY_EVENT       133
    KeyEvent        134     135     140     141     143     145     148     149     150
    LEFT_ALT_PRESSED        149
    LEFT_CTRL_PRESSED       150
    LPWORD           19     374     405     440     494     530
    RIGHT_ALT_PRESSED       149
    RIGHT_CTRL_PRESSED              150
    ReadConsoleInput        132
    ReadConsoleOutputAttribute               294     306     513
    ReadConsoleOutputCharacterA              283     305     512
    SHIFT_PRESSED           148
    SHORT           112     113
    STD_INPUT_HANDLE         33
    STD_OUTPUT_HANDLE        34
    SetConsoleCursorInfo             57     319     324     329     333
    SetConsoleCursorPosition        114
    WIN32_LEAN_AND_MEAN              13
    WORD             19     137     291     302     362     395
    WaitForSingleObject             130
    WriteConsoleOutputAttribute              367     387     399     430     456     556
    WriteConsoleOutputCharacterA             346     355     366     386     429     485     555
    X        80      99     112     281     292     303     344     353     363     380     396     427
           454     483     510     553
    Y        87     106     113     282     293     304     345     354     364     381     397     428
           455     484     511     554
    __CYGWIN32__             18     142
    attr             51      68     298     307     358     365     370     384     391     398     402
           422     436     450     562     567
    bKeyDown        134
    bVisible         59+     72+    317     322     327     332
    cci              56      57      58      59      65      71      72      73     312     313     317
           318     319     322     323     324     327     328     329     332     333
    ch              278     283     284     298     305     340     346     358     366     466     479
           562     567
    coord           280     281     282     283     290     292     293     294     301     303     304
           305     306     343     344     345     346     352     353     354     355     361     363
           364     366     367     373     380     381     386     387     394     396     397     399
           404     427     428     429     430     439     454     455     456     469     483     484
           485     493     510     511     512     513     529     553     554     555     556
    csbi             78      79      80      85      86      87      97      98      99     104     105
           106
    dest            490     516     517     518     519
    dwControlKeyState       148     149     150
    dwCursorPosition         99     106     111     112     113     114
    dwSize           58+     71+     80      87     318     323     328
    fAlt            138     149     222     240
    fCtrl           138     150     157     164     171     178     185     192     199     206     244
    fShift          138     148     226
    far              19
    free            388     417     432     433     458     487     505     522     523     541     558
           559
    h        10      11      15
    height           69
    i       372     382+    384     407     420+    422     423     438     448+    450     468     477+
           479     492     514+    516     518     528     546+    548     550
    iKey            119     235     242     246     250     256     258     261     266     271     273
    irBuffer        120     128+    132     133     134     135     140     141     143     145     148
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** by: Walter Bright via Usenet C newsgroup
**
** modified by: Bob Stout based on a recommendation by Ray Gardner
**
** modified by: David Gersic to deal with binary files
**
** There is no point in going to asm to get high speed file copies. Since it
** is inherently disk-bound, there is no sense (unless tiny code size is
** the goal). Here's a C version that you'll find is as fast as any asm code
** for files larger than a few bytes (the trick is to use large disk buffers):
*/

#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "snipfile.h"              /* Contains prototype for fdcopy()  */

#if !defined(__ZTC__) && !defined(__TURBOC__)
 #include <sys\types.h>
#endif

#include <sys\stat.h>

int file_append(char *from, char *to)
{
      int fdfrom,fdto;

      fdfrom = open(from,O_RDONLY|O_BINARY,0);
      if (fdfrom < 0)
            return 1;

      /* Open R/W by owner, R by everyone else        */

      fdto=open(to,O_BINARY|O_CREAT|O_APPEND|O_RDWR,S_IREAD|S_IWRITE);
      if (fdto >= 0)
      {
            if (Success_ == fdcopy(fdfrom, fdto))
            {
                  close(fdto);
                  close(fdfrom);
                  return Success_;
            }
            else
            {
                  close(fdto);
                  remove(to);             /* delete any partial file  */
            }
      }
      close(fdfrom);
      return Error_;
}
```

```
TEXT STATISTICS

  1508 characters
    54 lines

LEXICAL STATISTICS

     5 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     0 constants [character]
     1 constants [string]
     4 constants [numeric]

SYMBOL TABLE

    Error_          53
    O_BINARY|O_CREAT|O_APPEND|O_RDWR        37
    O_RDONLY|O_BINARY       31
    S_IREAD|S_IWRITE        37
    Success_        40      44
    __TURBOC__              21
    __ZTC__         21
    close           42      43      48      52
    defined         21+
    fcntl           18
    fdcopy          40
    fdfrom          29      31      32      40      43      52
    fdto            29      37      38      40      42      48
    file_append             27
    from            27      31
    h       16      17      18      22      25
    io              17
    open            31      37
    remove          49
    stdlib          16
    sys\stat        25
    sys\types       22
    to              27      37      49
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** by: Walter Bright via Usenet C newsgroup
**
** modified by: Bob Stout, Ray Gardner, and David Gersic
**
** There is no point in going to asm to get high speed file copies. Since it
** is inherently disk-bound, there is no sense (unless tiny code size is
** the goal). Here's a C version that you'll find is as fast as any asm code
** for files larger than a few bytes (the trick is to use large disk buffers):
*/

#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "snipfile.h"

#if !defined(__ZTC__) && !defined(__TURBOC__)
 #include <sys\types.h>
#endif

#include <sys\stat.h>

int file_copy(char *from, char *to)
{
      int fdfrom,fdto;

      fdfrom = open(from,O_RDONLY|O_BINARY,0);
      if (fdfrom < 0)
            return Error_;

      /* Open R/W by owner, R by everyone else        */

      fdto=open(to,O_BINARY|O_CREAT|O_TRUNC|O_RDWR,S_IREAD|S_IWRITE);
      if (fdto >= 0)
      {
            if (Success_ == fdcopy(fdfrom, fdto))
            {
                  close(fdto);
                  close(fdfrom);
                  return Success_;
            }
            else
            {
                  close(fdto);
                  remove(to);             /* delete any partial file  */
            }
      }
      close(fdfrom);
      return Error_;
}

int fdcopy(int fdfrom, int fdto)
{
      int bufsiz, retval = Error_;

      /* Use the largest buffer we can get    */

      for (bufsiz = 0x4000; bufsiz >= 128; bufsiz >>= 1)
      {
            register char *buffer;

            buffer = (char *) malloc(bufsiz);
            if (buffer)
            {
                  while (1)
                  {
                        register int n;

                        n = read(fdfrom,buffer,bufsiz);
                        if (n == -1)            /* if error             */
                              break;
                        if (n == 0)             /* if end of file       */
                        {
                              retval = Success_;
                              break;
                        }
                        if (n != write(fdto,buffer,(unsigned) n))
                              break;
                  }
                  free(buffer);
                  break;
            }
      }
      return retval;
}
```

TEXT STATISTICS

2424 characters
87 lines

LEXICAL STATISTICS

7 comments [std-C]
0 comments [C++]
8 preprocessor instructions
0 constants [character]
1 constants [string]
9 constants [numeric]

SYMBOL TABLE

Error_		31	51	56				
O_BINARY|O_CREAT|O_TRUNC|O_RDWR		35						
O_RDONLY|O_BINARY		29						
S_IREAD|S_IWRITE		35						
Success_		38	42	76				
__TURBOC__		19						
__ZTC__		19						
buffer		62	64	65	71	79	82	
bufsiz		56	60+	64	71			
close		40	41	46	50			
defined		19+						
fcntl		16						
fdcopy		38	54					
fdfrom		27	29	30	38	41	50	54	71
fdto		27	35	36	38	40	46	54	79
file_copy		25						
free		82						
from		25	29					
h	14	15	16	20	23			
io		15						
malloc		64						
n	69	71	72	74	79+			
open		29	35					
read		71						
remove		47						
retval		56	76	86				
stdlib		14						
sys\stat		23						
sys\types		20						
to		25	35	47				
write		79						

```
 1 | /* +++Date last modified: 05-Jul-1997 */
 2 |
 3 | /*
 4 |    File wc.c - a sample word count program
 5 |    Written and submitted to public domain by Jay Elkes
 6 |    April, 1992
 7 | */
 8 |
 9 | #include <stdio.h>
10 | #include <string.h>
11 | #include <ctype.h>
12 |
13 | int main (int argc, char *argv[])
14 | {
15 |       FILE *infileptr;
16 |       char infile[80];
17 |
18 |       long int nl = 0;
19 |       long int nc = 0;
20 |       long int nw = 0;
21 |
22 |       int state = 0;
23 |       const int  NEWLINE = '\n';
24 |       int  c;
25 |
26 | /*  The program name itself is the first command line argument so we
27 |     ignore it (argv[0]) when showing user entered parameters. */
28 |
29 |       switch (argc - 1)
30 |       {
31 |       case (0):
32 |             printf("no parameters\n");
33 |             return 12;
34 |       case (1):
35 |             break;
36 |       default:
37 |             printf("too many parameters\n");
38 |             return 12;
39 |       }
40 |
41 |       strcpy(infile,argv[1]);
42 |
43 |       infileptr = fopen(infile,"rb");
44 |       if (infileptr == NULL)
45 |       {
46 |             printf("Cannot open %s\n",infile);
47 |             return 12;
48 |       }
49 |
50 |       while ((c = getc(infileptr)) != EOF)
51 |       {
52 |             ++nc;
53 |             if (c == NEWLINE)
54 |                   ++nl;
55 |             if (isspace(c))
56 |                   state = 0;
57 |             else if (state == 0)
58 |             {
59 |                   state = 1;
60 |                   ++nw;
61 |             }
62 |       }
63 |
64 |       /* Final Housekeeping */
65 |
66 |       printf("%ld Lines, %ld Words, %ld Characters", nl, nw, nc);
67 |       return 0;
68 | }
```

```
TEXT STATISTICS

  1487 characters
    68 lines

LEXICAL STATISTICS

     4 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     1 constants [character]
     5 constants [string]
    16 constants [numeric]

SYMBOL TABLE

    EOF             50
    FILE            15
    NEWLINE         23      53
    NULL            44
    argc            13      29
    argv            13      41
    c       24      50      53      55
    ctype           11
    fopen           43
    getc            50
    h        9      10      11
    infile          16      41      43      46
    infileptr       15      43      44      50
    isspace         55
    main            13
    nc              19      52      66
    nl              18      54      66
    nw              20      60      66
    printf          32      37      46      66
    state           22      56      57      59
    stdio            9
    strcpy          41
    string          10
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #include<stdio.h>
  4 |
  5 | char *c[] = { "ENTER", "NEW", "POINT", "FIRST" };
  6 | char **cp[] = { c+3, c+2, c+1, c };
  7 | char ***cpp = cp;
  8 |
  9 | main()
 10 | {
 11 |     printf("%s", **++cpp);
 12 |     printf("%s ", *--*++cpp+3);
 13 |     printf("%s", *cpp[-2]+3);
 14 |     printf("%s\n", cpp[-1][-1]+1);
 15 |     return 0;
 16 | }
```

TEXT STATISTICS

```
   331 characters
    16 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
     1 preprocessor instructions
     0 constants [character]
     8 constants [string]
    10 constants [numeric]
```

SYMBOL TABLE

```
    c         5       6+
    cp                6       7
    cpp               7      11      12      13      14
    h         3
    main              9
    printf           11      12      13      14
    stdio             3
```

```
/* +++Date last modified: 05-Jul-1997 */

/*****************************************************************************
*     WHAT.c
*  @(#)
*     @(#) Scans binary or text files for a token and prints the line following
*     @(#) the token on stdout (normally the screen).
*     @(#) The token is by default @(#) like in the UNIX version of WHAT.
*  @(#)
*
*
*** CORRECTIONS ************************************************************
*1995-07-30/Bac:
*-    FindStringInFile() included explicit in project and recompiled.
*-    DEBUGTRACE inserted and iDebugFlag added
*1995-09-20/Bac:  v. 0.03
*-    fsif() ajusted of searching in last block of a file.
*1995-10-26/Bac: v. 0.04
*-    Rewritten for public use
*
*** BUGS & FLAWS ***********************************************************
*
*
*****************************************************************************
*@(#)1995 Erik Bachmann                                                      *
*
* Released to public domain 27-Oct-95
*****************************************************************************/
#include    "bacstd.h"                                      /* */
#include    <string.h>                                      /* strlen(),  */
#include    <stdio.h>                                       /* FILE */
#include    "dirport.h"


PROGRAMINF("WHAT", "0.04", "1995-06-18", "(p)1995 Erik Bachmann");

/*** MACROS ******************************************************************/
#define     MAXPATH     80
#define     MAXLINE     256

/* TRACE for debugging is NOT implemented in this version */
#define     TRACEFILE               NULL
#define     ON_DEBUG                NULL;
#define     TRACE                   _0
#define     DEBUGTRACE              _0

char  *pszErrMsg  = NULL;



/*** PROTOTYPES **************************************************************/
int main(int argc, char *argv[]);


/*** GLOBAL VARABLES *********************************************************/

static char szInputFile[MAXPATH];
static char szToken[MAXLINE] = "\x40(#)"; /* Token to identify comments */


const char  *HELPFORMAT =
{
      "Scans binary or text files for a token and prints the line following\n"
      "the token on stdout (normally the screen).\n"
      "The token is by default \x40(#) like in the UNIX version of WHAT.\n\n"
      "CALL: %Fs <filename> {/Ttoken}\n\n"
      "\tfilename\tfilename or wildcard\n"
      "\t/T\tToken if standard \x40(#) is not used.\n\n"
};

/*** FUNCTIONS ***************************************************************/
/* Dummy function */

int _0(char *fmt, ...)
{     }     /*** _0 ***/

/*
 /=====================================\
|      SEARCH_FILE                      |
|---------------------------------------|
|\=====================================/
|
|     Scanning through a binary file for a token. Prints the following line on
|  stdout.
|
|
|-------------------------------------------------------------------------------|
| CALL:
|    search_file(ffblk.ff_name);
|
| HEADER:
|    %
|
| GLOBALE VARIABLES:
|    szToken                 The Token to search for
|
| ARGUMENTS:
|    *szInputFile       Name of a file to scan for template strings
|
```

```
| |
| | PROTOTYPE:
| |    int  search_file(char *szInputFile)
| |
| | RETURN VALUE:
| |    int
| |
| | MODULE:
| |    what.c
| |
| |-KNOWN ERRORS AND FLAWS-------------------------------------------------|
| |
| |
| |
| |
| |-CORRECTIONS------------------------------------------------------------|
| |
| |
| |
| |
| |-----------------------------------------------------------------------|
| |1995-06-20/Erik Bachmann
| \=======================================================================|*/
|
| int  search_file(char *szInputFile)
| {
|       int         iStatusFlag       = 0;                    /* Return value */
|       int         iTokenLength      = 0;
|       char        szStr[MAXLINE+1];
|       long        lStartOffset      = 0L;
|
|       FILE        *fpInFile;
|
|       /*-------------------------------------*/
|
|       iTokenLength = strlen(szToken);
|
|       if (0 >= iTokenLength)
|       {
|             pszErrMsg   = "Token is too short (ie. < 1)";
|             fprintf(stderr, pszErrMsg);
|             DEBUGTRACE(TRACEFILE, pszErrMsg);
|             return(1);
|       }
|
|       if (0 == (fpInFile = fopen(szInputFile, "rb")))
|       {
|             fprintf(stderr, "Cannot open input file: %s", szInputFile);
|             return(1);
|       }
|
|       lStartOffset = (long) find_str_in_file(fpInFile, lStartOffset, szToken);
|
|       if (0 <= lStartOffset)
|       {
|             fprintf(stdout, "\n - %s\n", szInputFile);
|             /* First found -> write filename */
|       }
|
|       while (0 <= lStartOffset)
|       {
|             ON_DEBUG { TRACE(TRACEFILE, "\n\t%s found in %s offset %lu",
|                         szToken, szInputFile, lStartOffset); }
|
|             /* Write to file if debug flag in active */
|
|             lStartOffset += (long)strlen(szToken);
|
|             /* Go beyond token */
|
|             fseek(fpInFile, lStartOffset, SEEK_SET);
|
|             /* Go to start offset */
|
|             fgets(szStr, MAXLINE, fpInFile);
|
|             /* Get the string after token */
|
|             strtrimr(&szStr);                  /* Remove trailing line feed */
|
|             fprintf(stdout, "%s\n", szStr);
|
|             /* Write string */
|
|             lStartOffset += (long)strlen(szStr);
|
|             /* Next search starts from end of string */
|
|             lStartOffset = (long)find_str_in_file(fpInFile,
|                   (long)lStartOffset, szToken);
|
|             /* Find next token */
|       }
|       fclose(fpInFile);
|
|       return(iStatusFlag);                     /* Return default value */
| }
|
|
| /**** Main ************************************************************/
```

```
201 |
202 | /*
203 |  /=====================================\
204 |  |   MAIN
205 |  |-------------------------------------|
206 |  |\=====================================/
207 |  |
208 |  | Looping through files matching the first argument
209 |  |
210 |  |
211 |  |
212 |  |
213 |  | Virtual main()
214 |  |
215 |  | See template.c
216 |  |
217 |  |-KNOWN ERRORS AND FLAWS-----------------------------------------------------|
218 |  |
219 |  |
220 |  |
221 |  |
222 |  |-CORRECTIONS----------------------------------------------------------------|
223 |  |
224 |  |
225 |  |
226 |  |
227 |  |------------------------------------------------------------------------------|
228 |  |1995-06-19/Erik Bachmann
229 |  \==============================================================================|*/
230 |
231 |  int main(int argc, char *argv[])
232 |  {
233 |        DOSFileData       ffblk;                        /* File control block */
234 |        int                     done;                   /* Done flag */
235 |
236 |        /*--------------------------------*/
237 |
238 |        fprintf(stderr, "\n%Fs v. %Fs : %s\n", PROGNAME, PROGVER,
239 |                stModulInfo.pszCopyright);
240 |        fprintf(stdout, "Argument: %s\n", argv[1]);
241 |
242 |        if ('?' == argv[1][1])
243 |        {
244 |              fprintf(stderr, HELPFORMAT, argv[0]);
245 |              exit();
246 |        }
247 |        strcpy(szInputFile, argv[1]);
248 |
249 |        done = FIND_FIRST(szInputFile, 0, &ffblk);
250 |
251 |        /* Find first file */
252 |
253 |        while (!done)
254 |        {
255 |              search_file(ff_name(&ffblk));  /* Search file for tokens */
256 |              done = FIND_NEXT(&ffblk);      /* Find next matching file */
257 |        }
258 |
259 |        return(0);
260 |  }
```

TEXT STATISTICS

```
  7370 characters
   260 lines
```

LEXICAL STATISTICS

```
    34 comments [std-C]
     0 comments [C++]
    10 preprocessor instructions
     1 constants [character]
    21 constants [string]
    19 constants [numeric]
```

SYMBOL TABLE

```
    DEBUGTRACE              46     142
    DOSFileData            233
    FILE           132
    FIND_FIRST             249
    FIND_NEXT      256
    HELPFORMAT              62     244
    MAXLINE         40      59     129     175
    MAXPATH         39      58
    NULL            43      44      48
    ON_DEBUG        44     162
    PROGNAME       238
    PROGRAMINF              36
    PROGVER        238
    SEEK_SET       171
    TRACE           45     162
    TRACEFILE       43     142     162
    _0              45      46      75
    argc            53     231
    argv            53     231     240     242     244     247
    done           234     249     253     256
    exit           245
    fclose         194
    ff_name        255
    ffblk          233     249     255     256
    fgets          175
    find_str_in_file       152     189
    fmt             75
    fopen          146
    fpInFile       132     146     152     171     175     189     194
    fprintf        141     148     156     181     238     240     244
    fseek          171
    h       31      32
    iStatusFlag            127     196
    iTokenLength           128     136     138
    lStartOffset           130     152+    154     160     163     167     171     185     189     190
    main            53     231
    pszCopyright           239
    pszErrMsg       48     140     141     142
    search_file            125     255
    stModulInfo            239
    stderr         141     148     238     244
    stdio           32
    stdout         156     181     240
    strcpy         247
    string          31
    strlen         136     167     185
    strtrimr       179
    szInputFile             58     125     146     148     156     163     247     249
    szStr          129     175     179     181     185
    szToken         59     136     152     163     167     190
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  WHERE.C:  will search all DIRs on the given drive for specified file.
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>

#if defined(__ZTC__)
 #include <direct.h>
 #define GetDrive(d) dos_getdrive(&d)
 #define SetDrive(d) {unsigned x;dos_setdrive(d,&x);}
#elif defined(__TURBOC__)
 #include <dir.h>
 #define GetDrive(d) ((d) = getdisk() + 1)
 #define SetDrive(d) (setdisk(d - 1))
#else /* assume MSC */
 #include <direct.h>
 #define GetDrive(d) _dos_getdrive(&d)
 #define SetDrive(d) {unsigned x;_dos_setdrive(d,&x);}
#endif

#include "dirport.h"

int count=0;

main(int argc, char *argv[])
{
      char *curdir,
           sought[80],
          *temp;
      int   newdrive, p;
      void  searchdir(char *dir, char *ptrn);
      unsigned curdrive;

      /*  Find out where we are */

      curdir=getcwd(NULL,80);
      GetDrive(curdrive);

      /*  Find out what we're looking for  */

      if(argc>1)
            strcpy(sought,argv[1]);
      else
      {
            printf("\n\nPattern to search for: ");
            gets(sought);
      }

      /*  Get designator for another drive if specified  */

      if(sought[1]==':')
      {
            newdrive=(toupper(sought[0]))-64;       /* convert  */
            SetDrive(newdrive);
            p = (sought[2]=='\\') ? 3:2;
            strcpy(sought, &(sought[p]));
      }

      /*  Add wildcard prefix/suffix if necessary  */

      if(sought[0]=='.')
      {
            temp=strcat("*",sought);        /*  set prefix  */
            strcpy(sought,temp);
      }
      if(!strchr(sought,'.'))
            strcpy(sought,"*.*");           /*  set suffix  */

      /*  Perform search for pattern starting in root  */

      searchdir("\\",sought);
      printf("\nNumber of matches: %d",count);

      /*  Restore Original Drive and Directory  */

      SetDrive(curdrive);
      chdir(curdir);
      return EXIT_SUCCESS;
}

/*---------------------------------------------------------------------------*/

void searchdir(char *path, char *ptrn)
{
      DOSFileData   *f;
      char          *wholepath;
      unsigned       rtn;

      chdir(path);                        /*  change to new path  */
      wholepath = getcwd(NULL, 80);       /*  get full path name  */
      f = malloc(sizeof(*f));

      /*  Search for filename matches in this directory  */
```

```
 101 |
 102 |       rtn = FIND_FIRST(ptrn, _A_ANY, f);
 103 |       while (rtn == 0)
 104 |       {
 105 |             if (!(ff_attr(f) & _A_SUBDIR ))
 106 |                   printf("%s\\%s\n",wholepath,ff_name(f));
 107 |             else  printf("%s\\%s <DIR>\n", wholepath, ff_name(f));
 108 |             ++count;
 109 |
 110 |             rtn = FIND_NEXT(f);           /* find next match      */
 111 |       }  /* end while loop  */
 112 |
 113 |       /*  Now search any subdirectories under this directory  */
 114 |
 115 |       rtn = FIND_FIRST("*.*", _A_SUBDIR, f);
 116 |       while (rtn == 0)
 117 |       {
 118 |             if ((ff_attr(f) & _A_SUBDIR) && (ff_name(f)[0] != '.'))
 119 |             {
 120 |                   searchdir(ff_name(f), ptrn);  /* recursive call */
 121 |                   chdir(wholepath);
 122 |             }
 123 |             rtn = FIND_NEXT(f);           /* search next dir      */
 124 |       }
 125 |
 126 |       free(wholepath);
 127 |       free(f);
 128 | }
```

TEXT STATISTICS

```
  3392 characters
   128 lines
```

LEXICAL STATISTICS

```
    21 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
     5 constants [character]
     9 constants [string]
    18 constants [numeric]
```

SYMBOL TABLE

```
    DOSFileData             92
    EXIT_SUCCESS            85
    FIND_FIRST             102     115
    FIND_NEXT      110     123
    GetDrive        16      20      24      44
    NULL            43      97
    SetDrive        17      21      25      61      83
    _A_ANY         102
    _A_SUBDIR      105     115     118
    __TURBOC__              18
    __ZTC__         14
    _dos_getdrive           24
    _dos_setdrive           25
    argc            32      48
    argv            32      49
    chdir           84      96     121
    conio           10
    count           30      79     108
    ctype           12
    curdir          34      43      84
    curdrive        39      44      83
    d       16+     17+     20+     21+     24+     25+
    defined         14      18
    dir             19      38
    direct          15      23
    dos              9
    dos_getdrive            16
    dos_setdrive            17
    f       92      98+    102     105     106     107     110     115     118+    120     123     127
    ff_attr        105     118
    ff_name        106     107     118     120
    free           126     127
    getcwd          43      97
    getdisk         20
    gets            53
    h        7       8       9      10      11      12      15      19      23
    main            32
    malloc          98
    newdrive        37      60      61
    p       37      62      63
    path            90      96
    printf          52      79     106     107
    ptrn            38      90     102     120
    rtn             94     102     103     110     115     116     123
    searchdir       38      78      90     120
    setdisk         21
    sought          35      49      53      58      60      62      63+     68      70      71      73
            74      78
    stdio            7
    stdlib           8
    strcat          70
    strchr          73
    strcpy          49      63      71      74
    string          11
```

```
temp            36      70      71
toupper         60
wholepath       93      97      106     107     121     126
x       17+     25+
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  WORDWRAP.C - Simple CRT word wrap demonstration routine
**
**  public domain by Robert Morgan
*/

#include <stdio.h>
#include <conio.h>
#include <string.h>

int get_ln(int rmargin);
void clr_eol(const int curpos, const int pos);

main()
{
      printf("Enter text.  Press CTRL-A to quit.\n");
      while((get_ln(75)) != 0)      /* Change 75 to whatever number you */
            ;                       /* wish to be the right margin      */
      return 0;
}

void clr_eol(const int curpos, const int pos)
{
      int distance;
      int count;

      distance = curpos - pos;

      for (count = 1; count <= distance; count++)
            putch('\b');
      for (count = 1; count <= distance; count++)
            putch(' ');
}

int get_ln(int rmargin)
{
      char word[80];
      static int wordpos = 0;
      static int curpos = 1;
      static int ch = 0;
      static int pos = 0;

      word[wordpos] = '\0';

      while (ch != 1)
      {
            ch = getch();

            switch(ch)
            {
            case 1:
                  return(0);
            case ' ':
                  pos = curpos;
                  putch(' ');
                  curpos++;
                  wordpos = 0;
                  word[0] = '\0';
                  break;
            case '\b':
                  putch('\b');
                  curpos--;
                  if (wordpos > 0)
                        wordpos--;
                  break;
            case '\r':
                  puts("\r");
                  wordpos = 0;
                  word[wordpos] = '\0';
                  curpos = 1;
                  pos = 0;
                  break;
            default:
                  putch(ch);
                  word[wordpos] = (char)ch;
                  curpos++;
                  wordpos++;
                  break;
            }

            if(curpos == rmargin)
            {
                  word[wordpos] = '\0';
                  clr_eol(curpos,pos);
                  wordpos = 0;
                  curpos = strlen(word);
                  pos = 0;
                  puts("\r");
                  printf("%s",word);
            }
      }
      return -1;
}
```

TEXT STATISTICS

```
  2390 characters
    95 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
    11 constants [character]
     4 constants [string]
    22 constants [numeric]
```

SYMBOL TABLE

```
    ch              42      47      49      51      76      77
    clr_eol         14      24      86
    conio           10
    count           27      31+     33+
    curpos          14      24      29      41      56      58      64      72      78      83      86
          88
    distance        26      29      31      33
    get_ln          13      19      37
    getch           49
    h         9     10      11
    main            16
    pos             14      24      29      43      56      73      86      89
    printf          18      91
    putch           32      34      57      63      76
    puts            69      90
    rmargin         13      37      83
    stdio            9
    string          11
    strlen          88
    word            39      45      60      71      77      85      88      91
    wordpos         40      45      59      65      66      70      71      77      79      85      87
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  WPUTCH.C - Redirect text output to a scrollable window
**
**  public domain demo by Gaines Wright
**
**  Note: This code has been tested to work with Borland C++ 4.0+,
**        Microsoft C 7.0+, and Mix Power C 2.2.0+.
**
**        It will *not* work With Zortech C++ or Symantec C++ 6.x.
**
**        It compiles, but does not currently work correctly, with
**        Watcom C 10.0+ or Symantec C++ 7.0+ - this will be fixed ASAP.
*/

#include <dos.h>
#include <stdlib.h>

#if __POWERC | __TURBOC__
  #pragma option -N-
  #define IREGS unsigned r_bp, unsigned r_di, unsigned r_si, unsigned r_ds,\
                unsigned r_es, unsigned r_dx, unsigned r_cx, unsigned r_bx,\
                unsigned r_ax, unsigned r_ip, unsigned r_cs, unsigned r_flags
  #ifdef __cplusplus
    #define PAR ...
  #else
    #define PAR void
  #endif
  #ifdef __TURBOC__
       void interrupt far (*oldint29)(PAR);
  #else
    void interrupt (far *oldint29)(PAR);
  #endif
  #define ICAST (void interrupt (far*)(PAR))
#else
  #if (defined(__ZTC__) && !defined(__SC__)) || \
       (defined(__SC__) && __SC__ < 700)
    #error
  #endif

  #pragma check_stack(off)
  #pragma check_pointer(off)
  #define IREGS unsigned r_es, unsigned r_ds, unsigned r_di, unsigned r_si,\
                unsigned r_bp, unsigned r_sp, unsigned r_bx, unsigned r_dx,\
                unsigned r_cx, unsigned r_ax, unsigned r_ip, unsigned r_cs,\
                unsigned flags
  void (interrupt far *oldint29)();
  #if !defined(__WATCOMC__)
    #define REGS _REGS
  #endif
  #define ICAST (void (__interrupt far*)())
  #define getvect(a) _dos_getvect(a)
  #define setvect(a,b) _dos_setvect(a,b)
  #define interrupt __interrupt
#endif

#define SCRPOS(x,y) ((x)*2+(y)*160)        /* For upper left corner at 0,0. */
#define UP   6
#define DOWN 7

char xtop,ytop,xbot,ybot;                  /* Window coordinates. */
char attrib;                               /* Window attribute. */
union REGS iregs,oregs;
char far *topl,far *topr,far *ptr,far *fin;/* Window edge pointers */
char far *screen=(char far*)0xb8000000l;   /* Screen pointer. */

void scrollwin(char dir,char lines,char at,char yt,char xt,char yb,char xb)
{
      iregs.h.ah=dir;
      iregs.h.al=lines;
      iregs.h.bh=at;
      iregs.h.ch=yt;
      iregs.h.cl=xt;
      iregs.h.dh=yb;
      iregs.h.dl=xb;
      int86(0x10,&iregs,&oregs);
}


void setwindow(int xt,int yt,int xb,int yb,char a)
{
      xtop=xt;                             /* Set window coordinates. */
      ytop=yt;
      xbot=xb;
      ybot=yb;
      attrib=a;
      topl=screen+SCRPOS(xtop,ytop); /* Initialize pointers to window edges. */
      topr=screen+SCRPOS(xbot,ytop);
      fin=screen+SCRPOS(xbot,ybot);
      ptr=topl;
      scrollwin(UP,0,attrib,ytop,xtop,ybot,xbot);   /* Clears window */
}

void winputch(char c)                      /* Confines output to a window. */
{
      switch(c)
      {
      case '\r':
            return;
```

```
101 |
102 |       case '\n':
103 |             ptr=topr+1;
104 |             break;       /* Treat '\n' as "\r\n". */
105 |
106 |       case '\t':
107 |             ptr+=16;
108 |             break;          /* Expand tabs. */
109 |
110 |       default  :
111 |             *ptr++=c;
112 |             ptr++;
113 |             break;  /* Skip attribute byte. */
114 |       }
115 |       if(ptr>fin)                  /* If at bottom of window scroll up. */
116 |       {
117 |             scrollwin(UP,1,attrib,ytop,xtop,ybot,xbot);
118 |             ptr=topl;                     /* Reset back to left edge. */
119 |       }
120 |       else if(ptr>topr)                   /* Wrap around at window right edge. */
121 |       {
122 |             topl+=160;                    /* For a screen width of 80. */
123 |                   topr+=160;
124 |             ptr=topl;
125 |       }
126 | }
127 |
128 | #ifdef __TURBOC__
129 | #pragma argsused /* shut up 'argument xxx is never used' warnings */
130 | #endif
131 |
132 | void interrupt far newint29(IREGS)
133 | {
134 |       winputch((char)r_ax);
135 | }
136 |
137 | main(void)
138 | {
139 |       oldint29=getvect(0x29);
140 |       setvect(0x29,ICAST newint29);
141 |       setwindow(0,0,79,24,7);                   /* Just to clear the screen. */
142 |       setwindow(0,4,45,9,15+(1<<4));
143 |       system("dir");
144 |       setwindow(30,18,74,23,15+(1<<4));
145 |       system("dir");
146 |       setvect(0x29,oldint29);
147 |       return 0;
148 | }
```

```
TEXT STATISTICS

  4371 characters
   148 lines

LEXICAL STATISTICS

    20 comments [std-C]
     0 comments [C++]
    35 preprocessor instructions
     3 constants [character]
     2 constants [string]
    36 constants [numeric]

SYMBOL TABLE

    DOWN            60
    ICAST           35      52     140
    IREGS           22      44     132
    N       21
    PAR             26      28      31      33      35
    REGS            50      64
    SCRPOS          58      88      89      90
    UP              59      92     117
    _REGS           50
    __POWERC        20
    __SC__          37      38+
    __TURBOC__              20      30     128
    __WATCOMC__             49
    __ZTC__         37
    __cplusplus             25
    __interrupt             52      55
    _dos_getvect            53
    _dos_setvect            54
    a       53+     54+     81      87
    ah              70
    al              71
    argsused       129
    at              68      72
    attrib          63      87      92     117
    b       54+
    bh              72
    c       95      97     111
    ch              73
    check_pointer           43
    check_stack             42
    cl              74
    defined         37+     38      49
    dh              75
    dir             68      70
    dl              76
    dos             17
    far             31      33      35      48      52      65+     66+    132
    fin             65      90     115
    flags           47
    getvect         53     139
    h       17      18      70      71      72      73      74      75      76
    int86           77
    interrupt       31      33      35      48      55     132
    iregs           64      70      71      72      73      74      75      76      77
    lines           68      71
    main           137
    newint29       132     140
    off             42      43
    oldint29        31      33      48     139     146
    option          21
    oregs           64      77
    ptr             65      91     103     107     111     112     115     118     120     124
    r_ax            24      46     134
    r_bp            22      45
    r_bx            23      45
    r_cs            24      46
    r_cx            23      46
    r_di            22      44
    r_ds            22      44
    r_dx            23      45
    r_es            23      44
    r_flags         24
    r_ip            24      46
    r_si            22      44
    r_sp            45
    screen          66      88      89      90
    scrollwin       68      92     117
    setvect         54     140     146
    setwindow       81     141     142     144
    stdlib          18
    system         143     145
    topl            65      88      91     118     122     124
    topr            65      89     103     120     123
    winputch        95     134
    x       58+
    xb              68      76      81      85
    xbot            62      85      89      90      92     117
    xt              68      74      81      83
    xtop            62      83      88      92     117
    y       58+
    yb              68      75      81      86
    ybot            62      86      90      92     117
    yt              68      73      81      84
    ytop            62      84      88      89      92     117
```

```
/* +++Date last modified: 05-Jul-1997 */

/* w_wrap.c */

/*
**  This is an attempt at a useful word-wrap function.  It is given an array
**  of characters ( a string ) and it modifies the string, replacing any
**  new-lines found with spaces and placing new-lines where needed to make
**  lines of the width specified, placing them only where there was previously
**  white-space. ( ie. Words are not split across lines. )  At the present
**  time it is rather stupid. 1) It doesn't know enough to split a line at an
**  existing hyphen.  2) It has no clue about how to hyphenate words.  3) It
**  makes no attempt at dealing intelligently with a singly word longer than
**  the specified line length 4) It does not deal intelligently with multiple
**  spaces new-lines, etc. ( eg. has no clue of paragraph separation, etc. )
**  OTOH, it does deal well with unformatted, left justified text.
**
**  Tabs will be considered the size specified.  Note that word_wrap() does
**  not actually expand tabs.  This is only to inform it of the number of
**  spaces the output device will expand them to, so it will know how much
**  they expand a line.  The only time word_wrap does anything with tabs, is
**  if the tab size is set to zero, in which case each tab is replaced with a
**  single space character.  This often provides the most useful output, since
**  tabs will often be in the wrong places after re-formatting, and is
**  therefor the default.
**
**
**  Publicly available contents:
**
**      char *word_wrap(char *string,long line_len);
**          Does the actual word-wrapping, as described above;
**          Parameters:
**            string:     actual string to work with
**            line_len:   length of lines for output
**          Returns:        pointer to justified string.
**
**          void set_tab_size(int size);
**            Set the number of spaces that tabs will be expanded to on output
**            default tab size is zero. (each tab replaced with a space char )
**            word_wrap does not actually expand tabs.  This only lets it keep
**            track of how many spaces they take up.  If this is set to
**            zero, each tab will be replaced with a single space.
**
**  Other procedures:
**      int get_word(char *string);
**          returns the number of characters in the next word in string,
**          including leading white-space characters.
**
**  This compiles without warnings and runs with the following compilers:
**      MS Quick C 2.51:
**      Borland C++ 2.0:            either as C or C++
**      GNU C++ 1.39, DOS port:     either as C or C++
**  As far as I know, it uses only portable, standard C constructs.  It should
**  compile and run with little or no modification under nearly any C compiler
**  and environment.
**
**
**  This code was written Nov 16, 1991 by Jerry Coffin.
**  It is hereby placed in the public domain, for free use by any and
**  all who wish to do so, for any use, public, private, or commercial.
*/

#include <ctype.h>
#include "w_wrap.h"

static int tab_size = 0;                /* size to consider tabs as */

static size_t get_word(char *string);     /* returns size of next word*/

void set_tab_size(size_t size)
{
      tab_size = size;
}

char *word_wrap(char *string, size_t line_len)
{
      size_t len,                       /* length of current word */
             current_len = 0;           /* current length of line */
      size_t start_line = 0;            /* index of beginning if line */

      while (0 != (len = get_word(&string[current_len + start_line])))
      {
            if (current_len + len < line_len)
                  current_len += len;
            else
            {
                  string[start_line+current_len] = '\n';
                  start_line += current_len + 1;
                  current_len = 0;
            }
      }
      return string;
}

static size_t get_word(char *string)
{
      register int i = 0, word_len = 0;

      if (!string[0])
            return 0;
```

```
101 |       while (isspace(string[i]))
102 |       {
103 |             if ('\t' == string[i])
104 |             {
105 |                   if (0 == tab_size)
106 |                         string[i] = ' ';
107 |                   else  word_len += tab_size-1;
108 |             }
109 |             else if ('\n' == string[i])
110 |                   string[i]=' ';
111 |             word_len++;
112 |             i++;
113 |       }
114 |       while (string[i] && !isspace(string[i++]))
115 |             word_len++;
116 |       return word_len;
117 | }
118 |
119 | #ifdef TEST
120 |
121 | #include "w_wrap.h"
122 |
123 | main()
124 | {
125 |       char *string =
126 |             "This is a long line\nto be wrapped by the w_wrap function. "
127 |             "Hopefully, things will work correctly and it will be wrapped "
128 |             "between words.  On the other hand, maybe I should hope that it "
129 |             "doesn't work well so I will have an opportunity\nto learn more "
130 |             "about what I'm doing";
131 |
132 |       printf("Here's a string wrapped to 40 columns:\n\n%s\n\n",
133 |             word_wrap(string, 40));
134 |       printf("And here it's wrapped to 72:\n\n%s\n\n",
135 |             word_wrap(string,72));
136 |       return 0;
137 | }
138 |
139 | #endif /* TEST */
```

TEXT STATISTICS

```
  5249 characters
   139 lines
```

LEXICAL STATISTICS

```
     9 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     5 constants [character]
     9 constants [string]
    15 constants [numeric]
```

SYMBOL TABLE

```
    TEST           119
    ctype           63
    current_len              78      81      83      84      87      88      89
    get_word        68      81      95
    h       63
    i       97     101     103     106     109     110     112     114+
    isspace        101     114
    len             77      81      83      84
    line_len        75      83
    main           123
    printf         132     134
    set_tab_size            70
    size            70      72
    size_t          68      70      75      77      79      95
    start_line              79      81      87      88
    string          68      75      81      87      92      95      99     101     103     106     109
          110     114+    125     133     135
    tab_size        66      72     105     107
    word_len        97     107     111     115     116
    word_wrap       75     133     135
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /* w_wrap.h */
  4 | /* prototypes for the functions in w_wrap.c */
  5 |
  6 | #ifndef W_WRAP__H
  7 | #define W_WRAP__H
  8 |
  9 | #include <stddef.h>     /* For size_t     */
 10 | #include <stdio.h>      /* For FILE       */
 11 |
 12 | char *word_wrap(char *string, size_t line_len);
 13 | void set_tab_size(size_t size);
 14 | void center(FILE *file, char *string, size_t width);
 15 |
 16 | #endif /* W_WRAP__H */
```

TEXT STATISTICS

```
   406 characters
    16 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    FILE            14
    W_WRAP__H        6       7
    center          14
    file            14
    h        9      10
    line_len        12
    set_tab_size            13
    size            13
    size_t          12      13      14
    stddef           9
    stdio           10
    string          12      14
    width           14
    word_wrap       12
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *   X00API.C: X00 FOSSIL driver
 *
 *   Created by R.F. Pels.
 *   modified by Bob Stout
 *   Placed in the public domain by R.F. Pels.
 */

#include "x00api.h"
#include <dos.h>
#include "mk_fp.h"

static union  REGS  x00regs;
static struct SREGS x00sregs;
int                 x00error = 0;

#if defined(__cplusplus) && __cplusplus
 extern "C" {
#endif

#ifndef MK_FP
 #define MK_FP(seg,offset) \
        ((void _far *)(((unsigned long)(seg)<<16) | (unsigned)(offset)))
#endif

#if defined(__TURBOC__) || defined (__POWERC)
 #define PEEK(s,o) peek(s,o)
#else /* MSC or ZTC */
 #define PEEK(s,o) *((unsigned _far *)(MK_FP((s),(o))))
#endif

unsigned int  x00_detect(void)
{
        unsigned int segofx00;
        unsigned int ofsofx00;

        /* Peek in interrupt vector table for start of FOSSIL */

        ofsofx00 = PEEK(0, X00_VECTOR * 4);
        segofx00 = PEEK(0, (X00_VECTOR * 4) + 2);

        /* Peek in start of FOSSIL + X00_IDOFFSET */

        return (PEEK(segofx00, ofsofx00 + X00_IDOFFSET));
}

FOSSILSTATUS  x00_set(unsigned char set, PORT port)
{
        FOSSILSTATUS retval;

        x00regs.x.ax = SET_BAUDRATE | set;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        retval.statusword = x00regs.x.ax;
        return retval;
}

FOSSILSTATUS  x00_tx_char(unsigned char c, PORT port)
{
        FOSSILSTATUS retval;

        x00regs.x.ax = TX_CHAR | c;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        retval.statusword = x00regs.x.ax;
        return retval;
}

unsigned char  x00_rx_char(PORT port)
{
        x00regs.x.ax = RX_CHAR;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        return (unsigned char)(x00regs.x.ax & 0xff);
}

FOSSILSTATUS  x00_status(PORT port)
{
        FOSSILSTATUS retval;

        x00regs.x.ax = STATUS;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        retval.statusword = x00regs.x.ax;
        return retval;
}

FOSSILINIT    x00_init(PORT port, unsigned char far *ctlc_flagbyte)
{
        FOSSILINIT retval;

        x00regs.x.ax = INITIALIZE;
        if (ctlc_flagbyte != (unsigned char far *)0)
        {
                x00regs.x.dx = 0x00ff;
                x00regs.x.bx = 0x4F50;
                segread(&x00sregs);
                x00sregs.es  = FP_SEG(ctlc_flagbyte);
```

```
101 |                   x00regs.x.cx = FP_OFF(ctlc_flagbyte);
102 |           }
103 |           else
104 |           {
105 |                   x00regs.x.bx = 0x0000; /* in any case _NOT_ 0x4f50 */
106 |                   x00regs.x.dx = port;
107 |           }
108 |           int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
109 |           retval.result       = x00regs.x.ax;
110 |           retval.max_function = (unsigned char)(x00regs.x.bx & 0xff);
111 |           retval.revision     = (unsigned char)(x00regs.x.bx >> 8);
112 |           return retval;
113 | }
114 |
115 | void          x00_deinit(PORT port)
116 | {
117 |           x00regs.x.ax = DEINITIALIZE;
118 |           x00regs.x.dx = port;
119 |           int86(X00_VECTOR, &x00regs, &x00regs);
120 | }
121 |
122 | unsigned int  x00_raise_dtr(PORT port)
123 | {
124 |           unsigned int retval;
125 |
126 |           x00regs.x.ax = RAISE_DTR;
127 |           x00regs.x.dx = port;
128 |           int86(X00_VECTOR, &x00regs, &x00regs);
129 |           if ((x00regs.x.ax & 0x0001) == 1)
130 |           {
131 |                   retval = X00_DTR_HIGH;
132 |           }
133 |           else    retval = X00_DTR_LOW;
134 |           return retval;
135 | }
136 |
137 | unsigned int  x00_lower_dtr(PORT port)
138 | {
139 |           unsigned int retval;
140 |
141 |           x00regs.x.ax = LOWER_DTR;
142 |           x00regs.x.dx = port;
143 |           int86(X00_VECTOR, &x00regs, &x00regs);
144 |           if ((x00regs.x.ax & 0x0001) == 1)
145 |           {
146 |                   retval = X00_DTR_HIGH;
147 |           }
148 |           else    retval = X00_DTR_LOW;
149 |           return retval;
150 | }
151 |
152 | FOSSILSYSINFO x00_sysinfo(void)
153 | {
154 |           FOSSILSYSINFO retval;
155 |
156 |           x00regs.x.ax = GET_SYS_INFO;
157 |           int86(X00_VECTOR, &x00regs, &x00regs);
158 |           retval.tick_number        = (unsigned char)(x00regs.x.ax & 0xff);
159 |           retval.ticks_per_second   = (unsigned char)(x00regs.x.ax >> 8);
160 |           retval.approx_ms_per_tick = x00regs.x.dx;
161 |           return retval;
162 | }
163 |
164 | void          x00_flush_output(PORT port)
165 | {
166 |           x00regs.x.ax = FLUSH_OUTPUT;
167 |           x00regs.x.dx = port;
168 |           int86(X00_VECTOR, &x00regs, &x00regs);
169 | }
170 |
171 | void          x00_purge_output(PORT port)
172 | {
173 |           x00regs.x.ax = PURGE_OUTPUT;
174 |           x00regs.x.dx = port;
175 |           int86(X00_VECTOR, &x00regs, &x00regs);
176 | }
177 |
178 | void          x00_purge_input(PORT port)
179 | {
180 |           x00regs.x.ax = PURGE_INPUT;
181 |           x00regs.x.dx = port;
182 |           int86(X00_VECTOR, &x00regs, &x00regs);
183 | }
184 |
185 | unsigned int  x00_tx_char_nowait(unsigned char c, PORT port)
186 | {
187 |           unsigned int retval;
188 |
189 |           x00regs.x.ax = TX_CHAR_NOWAIT | c;
190 |           x00regs.x.dx = port;
191 |           int86(X00_VECTOR, &x00regs, &x00regs);
192 |           if ((x00regs.x.ax & 0x0001) == 1)
193 |           {
194 |                   retval = X00_OK;
195 |           }
196 |           else    retval = X00_CHAR_NOT_SENT;
197 |           return retval;
198 | }
199 |
200 | unsigned int  x00_peek_ahead_input(PORT port)
```

```
{
        x00regs.x.ax = PEEK_AHEAD_INPUT;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        return x00regs.x.ax;
}

unsigned int   x00_peek_ahead_kbd(void)
{
        x00regs.x.ax = PEEK_AHEAD_KBD;
        int86(X00_VECTOR, &x00regs, &x00regs);
        return x00regs.x.ax;
}

unsigned int  x00_read_kbd(void)
{
        x00regs.x.ax = READ_KBD;
        int86(X00_VECTOR, &x00regs, &x00regs);
        return x00regs.x.ax;
}

void          x00_flow_control(FOSSILFLOWCTRL f, PORT port)
{
        x00regs.x.ax = FLOW_CONTROL | 0xf0 | f.flowword;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

unsigned int  x00_ctlc_ctlk_check(FOSSILCTLCCTLK c, PORT port)
{
        x00regs.x.ax = CTLC_CTLK_CHECK | c.checkword;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
        return x00regs.x.ax;
}

void          x00_set_cup(unsigned char row, unsigned char col)
{
        x00regs.x.ax = SET_CUP;
        x00regs.x.dx = (row << 8) | col;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

void          x00_get_cup(unsigned char *row, unsigned char *col)
{
        x00regs.x.ax = GET_CUP;
        int86(X00_VECTOR, &x00regs, &x00regs);
        *col = (unsigned char)(x00regs.x.dx & 0xff);
        *row = (unsigned char)(x00regs.x.dx >> 8);
}

void          x00_write_ANSI_char(unsigned char c)
{
        x00regs.x.ax = WRITE_ANSI_CHAR | c;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

void          x00_enable_watchdog(PORT port)
{
        x00regs.x.ax = ENABLE_WATCHDOG;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

void          x00_disable_watchdog(PORT port)
{
        x00regs.x.ax = DISABLE_WATCHDOG;
        x00regs.x.dx = port;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

void          x00_write_BIOS_char(unsigned char c)
{
        x00regs.x.ax = WRITE_BIOS_CHAR | c;
        int86(X00_VECTOR, &x00regs, &x00regs);
}

unsigned int  x00_insert_tick_func(void (far *tickfunc)())
{
        unsigned int retval;

        x00regs.x.ax = INSERT_TICK_FUNC;
        x00regs.x.dx = FP_OFF(tickfunc);
        segread(&x00sregs);
        x00sregs.es  = FP_SEG(tickfunc);
        int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
        if (x00regs.x.ax == 0x0000)
        {
                retval = X00_OK;
        }
        else    retval = X00_INS_TICK;
        return retval;
}

unsigned int  x00_delete_tick_func(void (far *tickfunc)())
{
        unsigned int retval;

        x00regs.x.ax = DELETE_TICK_FUNC;
        x00regs.x.dx = FP_OFF(tickfunc);
```

```
301 |         segread(&x00sregs);
302 |         x00sregs.es  = FP_SEG(tickfunc);
303 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
304 |         if (x00regs.x.ax == 0x0000)
305 |         {
306 |                 retval = X00_OK;
307 |         }
308 |         else    retval = X00_DEL_TICK;
309 |         return retval;
310 | }
311 |
312 | void          x00_boot_machine(unsigned int boottype)
313 | {
314 |         x00regs.x.ax = BOOT_MACHINE | (boottype & 0x0001);
315 |         int86(X00_VECTOR, &x00regs, &x00regs);
316 | }
317 |
318 | unsigned int  x00_read_block(unsigned int count, void far *buf, PORT port)
319 | {
320 |         x00regs.x.ax = READ_BLOCK;
321 |         x00regs.x.cx = count;
322 |         x00regs.x.dx = port;
323 |         segread(&x00sregs);
324 |         x00sregs.es  = FP_SEG(buf);
325 |         x00regs.x.di = FP_OFF(buf);
326 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
327 |         return x00regs.x.ax;
328 | }
329 |
330 | unsigned int  x00_write_block(unsigned int count, void far *buf, PORT port)
331 | {
332 |         x00regs.x.ax = WRITE_BLOCK;
333 |         x00regs.x.cx = count;
334 |         x00regs.x.dx = port;
335 |         segread(&x00sregs);
336 |         x00sregs.es  = FP_SEG(buf);
337 |         x00regs.x.di = FP_OFF(buf);
338 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
339 |         return x00regs.x.ax;
340 | }
341 |
342 | void          x00_start_break_signal(PORT port)
343 | {
344 |         x00regs.x.ax = START_BREAK_SIGNAL;
345 |         x00regs.x.dx = port;
346 |         int86(X00_VECTOR, &x00regs, &x00regs);
347 | }
348 |
349 | void          x00_stop_break_signal(PORT port)
350 | {
351 |         x00regs.x.ax = STOP_BREAK_SIGNAL;
352 |         x00regs.x.dx = port;
353 |         int86(X00_VECTOR, &x00regs, &x00regs);
354 | }
355 |
356 | unsigned int  x00_get_driverinfo(void far *buf, PORT port)
357 | {
358 |         x00regs.x.ax = GET_DRIVER_INFO;
359 |         x00regs.x.cx = sizeof(FOSSILINFO);
360 |         segread(&x00sregs);
361 |         x00sregs.es  = FP_SEG(buf);
362 |         x00regs.x.di = FP_OFF(buf);
363 |         x00regs.x.dx = port;
364 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
365 |         return x00regs.x.ax;
366 | }
367 |
368 | unsigned int  x00_install_appendage(unsigned char appcode,
369 |                                                 void (far *appfunc)())
370 | {
371 |         unsigned int retval;
372 |
373 |         x00regs.x.ax = INSTALL_APPENDAGE | appcode;
374 |         segread(&x00sregs);
375 |         x00sregs.es  = FP_SEG(appfunc);
376 |         x00regs.x.dx = FP_OFF(appfunc);
377 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
378 |         if (x00regs.x.ax == X00_PRESENT)
379 |         {
380 |                 if ((x00regs.x.bx & 0xff00) == 1)
381 |                 {
382 |                         retval = X00_OK;
383 |                 }
384 |                 else    retval = X00_INS_APP;
385 |         }
386 |         else    retval = X00_NOT_HERE;
387 |         return retval;
388 | }
389 |
390 | unsigned int  x00_remove_appendage(unsigned char appcode,
391 |                                                 void (far *appfunc)())
392 | {
393 |         unsigned int retval;
394 |
395 |         x00regs.x.ax = REMOVE_APPENDAGE | appcode;
396 |         segread(&x00sregs);
397 |         x00sregs.es  = FP_SEG(appfunc);
398 |         x00regs.x.dx = FP_OFF(appfunc);
399 |         int86x(X00_VECTOR, &x00regs, &x00regs, &x00sregs);
400 |         if (x00regs.x.ax == X00_PRESENT)
```

```
401 |         {
402 |                 if ((x00regs.x.bx & 0xff00) == 1)
403 |                 {
404 |                         retval = X00_OK;
405 |                 }
406 |                 else    retval = X00_REM_APP;
407 |         }
408 |         else    retval = X00_NOT_HERE;
409 |         return retval;
410 | }
411 |
412 | #if defined(__cplusplus) && __cplusplus
413 |  }
414 | #endif
```

TEXT STATISTICS

```
 11384 characters
   414 lines
```

LEXICAL STATISTICS

```
     6 comments [std-C]
     0 comments [C++]
    15 preprocessor instructions
     0 constants [character]
     3 constants [string]
    33 constants [numeric]
```

SYMBOL TABLE

```
    BOOT_MACHINE           314
    CTLC_CTLK_CHECK        231
    DEINITIALIZE           117
    DELETE_TICK_FUNC       299
    DISABLE_WATCHDOG       267
    ENABLE_WATCHDOG        260
    FLOW_CONTROL           224
    FLUSH_OUTPUT           166
    FOSSILCTLCCTLK         229
    FOSSILFLOWCTRL         222
    FOSSILINFO             359
    FOSSILINIT              90      92
    FOSSILSTATUS            49      51      60      62      79      81
    FOSSILSYSINFO          152     154
    FP_OFF         101     283     300     325     337     362     376     398
    FP_SEG         100     285     302     324     336     361     375     397
    GET_CUP        246
    GET_DRIVER_INFO        358
    GET_SYS_INFO           156
    INITIALIZE              94
    INSERT_TICK_FUNC       282
    INSTALL_APPENDAGE      373
    LOWER_DTR      141
    MK_FP           23      24      31
    PEEK            29      31      41      42      46
    PEEK_AHEAD_INPUT       202
    PEEK_AHEAD_KBD         210
    PORT            49      60      71      79      90     115     122     137     164     171     178
           185     200     222     229     258     265     318     330     342     349     356
    PURGE_INPUT            180
    PURGE_OUTPUT           173
    RAISE_DTR      126
    READ_BLOCK             320
    READ_KBD       217
    REGS            15
    REMOVE_APPENDAGE       395
    RX_CHAR         73
    SET_BAUDRATE            53
    SET_CUP        239
    SREGS           16
    START_BREAK_SIGNAL             344
    STATUS          83
    STOP_BREAK_SIGNAL      351
    TX_CHAR         64
    TX_CHAR_NOWAIT         189
    WRITE_ANSI_CHAR        254
    WRITE_BIOS_CHAR        274
    WRITE_BLOCK            332
    X00_CHAR_NOT_SENT      196
    X00_DEL_TICK           308
    X00_DTR_HIGH           131     146
    X00_DTR_LOW            133     148
    X00_IDOFFSET            46
    X00_INS_APP            384
    X00_INS_TICK           291
    X00_NOT_HERE           386     408
    X00_OK         194     289     306     382     404
    X00_PRESENT            378     400
    X00_REM_APP            406
    X00_VECTOR              41      42      55      66      75      85     108     119     128     143
           157     168     175     182     191     204     211     218     226     233     241     247
           255     262     269     275     286     303     315     326     338     346     353     364
           377     399
    __POWERC        28
    __TURBOC__              28
    __cplusplus             19+    412+
    _far            25      31
    appcode        368     373     390     395
    appfunc        369     375     376     391     397     398
    approx_ms_per_tick             160
    ax              53      56      64      67      73      76      83      86      94     109     117
           126     129     141     144     156     158     159     166     173     180     189     192
           202     205     210     212     217     219     224     231     234     239     246     254
           260     267     274     282     287     299     304     314     320     327     332     339
           344     351     358     365     373     378     395     400
    boottype       312     314
    buf            318     324     325     330     336     337     356     361     362
    bx              98     105     110     111     380     402
    c       60      64     185     189     229     231     252     254     272     274
    checkword      231
    col            237     240     244     248
    count          318     321     330     333
    ctlc_flagbyte           90      95     100     101
    cx             101     321     333     359
```

397 399

```
/* +++Date last modified: 05-Jul-1997 */

/*
 *  X00API.H - X00 FOSSIL driver HLLAPI
 *
 *   Created by R.F. Pels.
 *   modified by Bob Stout
 *   Placed in the public domain by R.F. Pels.
 */

#ifndef __X00API_H                      /*  Prevent double inclusion  */
#define __X00API_H

/*  FOSSIL specifics */

#define X00_VECTOR          0x14
#define X00_IDOFFSET        6

#if defined(__TURBOC__) || defined(__POWERC)
 #define _far far
#endif

/*  FOSSIL function request codes */

#define SET_BAUDRATE        0x0000
#define TX_CHAR             0x0100
#define RX_CHAR             0x0200
#define STATUS              0x0300
#define INITIALIZE          0x0400
#define DEINITIALIZE        0x0500
#define RAISE_DTR           0x0601
#define LOWER_DTR           0x0600
#define GET_SYS_INFO        0x0700
#define FLUSH_OUTPUT        0x0800
#define PURGE_OUTPUT        0x0900
#define PURGE_INPUT         0x0A00
#define TX_CHAR_NOWAIT      0x0B00
#define PEEK_AHEAD_INPUT    0x0C00
#define PEEK_AHEAD_KBD      0x0D00
#define READ_KBD            0x0E00
#define FLOW_CONTROL        0x0F00
#define CTLC_CTLK_CHECK     0x1000
#define SET_CUP             0x1100
#define GET_CUP             0x1200
#define WRITE_ANSI_CHAR     0x1300
#define ENABLE_WATCHDOG     0x1401
#define DISABLE_WATCHDOG    0x1400
#define WRITE_BIOS_CHAR     0x1500
#define INSERT_TICK_FUNC    0x1601
#define DELETE_TICK_FUNC    0x1600
#define BOOT_MACHINE        0x1700
#define READ_BLOCK          0x1800
#define WRITE_BLOCK         0x1900
#define START_BREAK_SIGNAL  0x1A01
#define STOP_BREAK_SIGNAL   0x1A00
#define GET_DRIVER_INFO     0x1B00
#define INSTALL_APPENDAGE   0x7e00
#define REMOVE_APPENDAGE    0x7f00

/* port numbers and variable type of portnumber in calls */

#define PORTCOM1            0
#define PORTCOM2            1
#define PORTCOM3            2
#define PORTCOM4            3
#define PORTCOM5            4
#define PORTCOM6            5
#define PORTCOM7            6
#define PORTCOM8            7
#define PORTSPECIAL         0x00ff

typedef unsigned int PORT;

/* defines components of baud settings call */

#define BAUD300             0x40
#define BAUD600             0x60
#define BAUD1200            0x80
#define BAUD2400            0xa0
#define BAUD4800            0xc0
#define BAUD9600            0xe0
#define BAUD19200           0x00
#define BAUD38400           0x20
#define PARITYNONE          0x00
#define PARITYODD           0x08
#define PARITYNONEALT       0x10
#define PARITYEVEN          0x18
#define STOP1BIT            0x00
#define STOP2BIT            0x04
#define WORD5BIT            0x00
#define WORD6BIT            0x01
#define WORD7BIT            0x02
#define WORD8BIT            0x03
#define FIDOSETTING         PARITYNONE | STOP1BIT | WORD8BIT
#define OPUSSETTING         PARITYNONE | STOP1BIT | WORD8BIT
#define SEADOGSETTING       PARITYNONE | STOP1BIT | WORD8BIT

/* Error numbers */

#define X00_OK              0
```

```
101 | #define X00_NOT_HERE        100
102 | #define X00_CHAR_NOT_SENT   101
103 | #define X00_NO_INPUT        0xffff
104 | #define X00_NO_KEY          0xffff
105 | #define X00_INS_TICK        104
106 | #define X00_DEL_TICK        105
107 | #define X00_INS_APP         106
108 | #define X00_REM_APP         107
109 | #define X00_DTR_HIGH        108
110 | #define X00_DTR_LOW         109
111 |
112 |
113 | /* FOSSIL initcall result type */
114 |
115 | #define X00_PRESENT         0x1954
116 |
117 | typedef struct {
118 |         unsigned int result;
119 |         unsigned char max_function;
120 |         unsigned char revision;
121 |         } FOSSILINIT;
122 |
123 | /* FOSSIL status return type: all fields are 1 if condition exists */
124 |
125 | typedef union {
126 |         struct STATUSBITS {
127 |             unsigned                  : 3;
128 |             unsigned alwayshigh       : 1;
129 |             unsigned                  : 3;
130 |             unsigned carrier_detect   : 1;
131 |             unsigned chars_in_input   : 1;
132 |             unsigned input_overrun    : 1;
133 |             unsigned                  : 3;
134 |             unsigned output_not_full  : 1;
135 |             unsigned output_empty     : 1;
136 |             unsigned                  : 1;
137 |             } statusbits;
138 |       unsigned int statusword;
139 |       } FOSSILSTATUS;
140 |
141 | /* FOSSIL info type */
142 |
143 | typedef struct {
144 |         unsigned int  size;
145 |         unsigned char version;
146 |         unsigned char revision;
147 |         unsigned int  ofs_fossil_id;
148 |         unsigned int  seg_fossil_id;
149 |         unsigned int  input_size;
150 |         unsigned int  input_avail;
151 |         unsigned int  output_size;
152 |         unsigned int  output_avail;
153 |         unsigned char screen_width;
154 |         unsigned char screen_height;
155 |         unsigned char baud_rate_mask;
156 |         } FOSSILINFO;
157 |
158 | /* FOSSIL system info type */
159 |
160 | typedef struct {
161 |         unsigned char tick_number;
162 |         unsigned char ticks_per_second;
163 |         unsigned int  approx_ms_per_tick;
164 |         } FOSSILSYSINFO;
165 |
166 | /* FOSSIL flow control type */
167 |
168 | typedef union {
169 |         struct FLOWBITS {
170 |             unsigned  xonxoff         : 1;
171 |             unsigned  ctsrts          : 1;
172 |             unsigned  remote_xonxoff  : 1;
173 |             unsigned                  : 5;
174 |             } flowbits;
175 |       unsigned char flowword;
176 |       } FOSSILFLOWCTRL;
177 |
178 | /* FOSSIL checks control type */
179 |
180 | #define X00_CTLCK           0x0001
181 | #define X00_NO_CTLCK        0x0000
182 |
183 | typedef union {
184 |         struct CHECKBITS {
185 |             unsigned  ctlc_ctlk         : 1;
186 |             unsigned  stop_transmitter  : 1;
187 |             unsigned                    : 6;
188 |             } checkbits;
189 |       unsigned char checkword;
190 |       } FOSSILCTLCCTLK;
191 |
192 | /* Macros to use with x00_boot_machine() */
193 |
194 | #define COLDBOOT        0
195 | #define WARMBOOT        1
196 |
197 | /* Macros to use with x00_read_kbd() */
198 |
199 | #define ISFNKEY(x)      (((x) & 0x00FF) == 0)
200 | #define FNKEYCODE(x)    ((x) >> 8)
```

```
201 |
202 | #if defined(__cplusplus) && __cplusplus
203 |  extern "C" {
204 | #endif
205 |
206 | unsigned int  x00_detect(void);
207 | FOSSILSTATUS  x00_set(unsigned char set, PORT port);
208 | FOSSILSTATUS  x00_tx_char(unsigned char c, PORT port);
209 | unsigned char x00_rx_char(PORT port);
210 | FOSSILSTATUS  x00_status(PORT port);
211 | FOSSILINIT    x00_init(PORT port, unsigned char _far *ctlc_flagbyte);
212 | void          x00_deinit(PORT port);
213 | unsigned int  x00_raise_dtr(PORT port);
214 | unsigned int  x00_lower_dtr(PORT port);
215 | FOSSILSYSINFO x00_sysinfo(void);
216 | void          x00_flush_output(PORT port);
217 | void          x00_purge_output(PORT port);
218 | void          x00_purge_input(PORT port);
219 | unsigned int  x00_tx_char_nowait(unsigned char c, PORT port);
220 | unsigned int  x00_peek_ahead_input(PORT port);
221 | unsigned int  x00_peek_ahead_kbd(void);
222 | unsigned int  x00_read_kbd(void);
223 | void          x00_flow_control(FOSSILFLOWCTRL f, PORT port);
224 | unsigned int  x00_ctlc_ctlk_check(FOSSILCTLCCTLK c, PORT port);
225 | void          x00_set_cup(unsigned char row, unsigned char col);
226 | void          x00_get_cup(unsigned char *row, unsigned char *col);
227 | void          x00_write_ANSI_char(unsigned char c);
228 | void          x00_enable_watchdog(PORT port);
229 | void          x00_disable_watchdog(PORT port);
230 | void          x00_write_BIOS_char(unsigned char c);
231 | unsigned int  x00_insert_tick_func(void (_far *tickfunc)());
232 | unsigned int  x00_delete_tick_func(void (_far *tickfunc)());
233 | void          x00_boot_machine(unsigned int boottype);
234 | unsigned int  x00_read_block(unsigned int count, void _far *buf, PORT port);
235 | unsigned int  x00_write_block(unsigned int count, void _far *buf,
236 | PORT port);
237 | void          x00_start_break_signal(PORT port);
238 | void          x00_stop_break_signal(PORT port);
239 | unsigned int  x00_get_driverinfo(void _far *buf, PORT port);
240 | unsigned int  x00_install_appendage(unsigned char appcode,
241 |                   void (_far *appfunc)());
242 | unsigned int  x00_remove_appendage(unsigned char appcode,
243 |                   void (_far *appfunc)());
244 |
245 | #if defined(__cplusplus) && __cplusplus
246 |  }
247 | #endif
248 |
249 | #endif /* __X00API_H */
```

TEXT STATISTICS

7952 characters
249 lines

LEXICAL STATISTICS

17 comments [std-C]
0 comments [C++]
94 preprocessor instructions
0 constants [character]
1 constants [string]
99 constants [numeric]

SYMBOL TABLE

BAUD1200 78
BAUD19200 82
BAUD2400 79
BAUD300 76
BAUD38400 83
BAUD4800 80
BAUD600 77
BAUD9600 81
BOOT_MACHINE 51
CHECKBITS 184
COLDBOOT 194
CTLC_CTLK_CHECK 42
DEINITIALIZE 30
DELETE_TICK_FUNC 50
DISABLE_WATCHDOG 47
ENABLE_WATCHDOG 46
FIDOSETTING 94
FLOWBITS 169
FLOW_CONTROL 41
FLUSH_OUTPUT 34
FNKEYCODE 200
FOSSILCTLCCTLK 190 224
FOSSILFLOWCTRL 176 223
FOSSILINFO 156
FOSSILINIT 121 211
FOSSILSTATUS 139 207 208 210
FOSSILSYSINFO 164 215
GET_CUP 44
GET_DRIVER_INFO 56
GET_SYS_INFO 33
INITIALIZE 29
INSERT_TICK_FUNC 49
INSTALL_APPENDAGE 57
ISFNKEY 199
LOWER_DTR 32
OPUSSETTING 95
PARITYEVEN 87
PARITYNONE 84 94 95 96
PARITYNONEALT 86
PARITYODD 85
PEEK_AHEAD_INPUT 38
PEEK_AHEAD_KBD 39
PORT 72 207 208 209 210 211 212 213 214 216 217
218 219 220 223 224 228 229 234 236 237 238 239
PORTCOM1 62
PORTCOM2 63
PORTCOM3 64
PORTCOM4 65
PORTCOM5 66
PORTCOM6 67
PORTCOM7 68
PORTCOM8 69
PORTSPECIAL 70
PURGE_INPUT 36
PURGE_OUTPUT 35
RAISE_DTR 31
READ_BLOCK 52
READ_KBD 40
REMOVE_APPENDAGE 58
RX_CHAR 27
SEADOGSETTING 96
SET_BAUDRATE 25
SET_CUP 43
START_BREAK_SIGNAL 54
STATUS 28
STATUSBITS 126
STOP1BIT 88 94 95 96
STOP2BIT 89
STOP_BREAK_SIGNAL 55
TX_CHAR 26
TX_CHAR_NOWAIT 37
WARMBOOT 195
WORD5BIT 90
WORD6BIT 91
WORD7BIT 92
WORD8BIT 93 94 95 96
WRITE_ANSI_CHAR 45
WRITE_BIOS_CHAR 48
WRITE_BLOCK 53
X00_CHAR_NOT_SENT 102
X00_CTLCK 180
X00_DEL_TICK 106
X00_DTR_HIGH 109
X00_DTR_LOW 110

```
    x00_write_block         235
    xonxoff         170
```

```
/* +++Date last modified: 05-Jul-1997 */

/*  xfile.c -- implementation for fast line buffered files
**
**  Currently (Sat  06-15-1991) XFILEs are for reading CR-LF terminated lines
**  from MS-DOS text files.  Period.  It's not that the method can't be used
**  as well for output buffering, or (in some form) for binary files, it's
**  that such are handled fast enough to suit me already, whereas text mode
**  input performance leaves me wishing for more speed.  This attempts to
**  solve that problem.
**
**  Sun  06-16-1991 -- CR-LF accepted, but so is bare LF now; the extracted
**  line does NOT have a NEWLINE at the end anymore (which will likely be
**  a mixed blessing...)
**
**  The code should be fairly portable: if/when I get around to polishing it
**  (and that won't be until I've used it some and am sure it's stable) I'll
**  be aiming for near-ANSI portability; for now I'm not pushing so very hard
**  for that.
**
**  The semantics are a bit odd: the lines are returned in a buffer that the
**  XFILE owns, and may be altered by a call to xgetline or xclose.  For
**  applications that resent this, XFILEs probably aren't a big win anyway,
**  but there might be some cases where using XFILE and copying (some) lines
**  is still a good idea.  The performance with long lines is good: it can
**  handle lines the size of the buffer, though it may truncate up to one
**  QUANTUM less one bytes "early": this depends on the location of the start
**  of the line in the buffer when we begin scanning.  In practice, XBUFSIZE
**  is probably larger than you'd set for a line buffer size anyway...
**
**  INTERNALS:
**
**  Reading the first buffer's worth at open time makes the EOF case easier to
**  detect.
**
**  TO DO:
**
**  clean up xgetline!
*/

#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "xfile.h"

#if !defined(__ZTC__) && !defined(__TURBOC__)
 static int DOS_OPEN(const char *name, int mode, ...)
 {
       int hdl;

       if (0 == _dos_open(name, mode, &hdl))
             return hdl;
       else  return -1;
 }

 static int READ(int fd, void *buf, size_t len)
 {
       unsigned count;

       if (0 == _dos_read(fd, buf, len, &count))
             return count;
       else  return -1;
 }
#endif

#ifndef XBUFN               /* set default # of quanta in buffer, allow -D */
 #define XBUFN 8
#endif

#define QUANTUM 512
#define XBUFSIZE (XBUFN * QUANTUM)


/*  xopen -- allocate and open an XFILE
**
**  NB: currently I'm designing these for READ-ONLY TEXT FILES only: the xopen
**      interface may have to be changed...
**
**  returns pointer to XFILE of opened file or null pointer on error
**
**  ? should it leave a better error description somewhere ?
*/

XFILE *xopen(char const *name)
{
      XFILE *f = malloc(sizeof(XFILE) + XBUFSIZE + 1);
      int n;

      if (f == NULL)
            goto error0;
      f->buf = (char *)f + sizeof(XFILE);

      if ((f->fd = DOS_OPEN(name, O_RDONLY)) < 0)
            goto error1;

      if ((n = READ(f->fd, f->buf, XBUFSIZE)) < 0)
            goto error2;

      f->buf[n] = 0;
      f->nextChar = f->buf;
```

```
101 |       return f;
102 |
103 | error2:
104 |       CLOSE(f->fd);
105 | error1:
106 |       free(f);
107 | error0:
108 |       return NULL;
109 | }
110 |
111 |
112 | /*
113 | **  xclose -- close and deallocate an XFILE
114 | */
115 |
116 | void xclose(XFILE *f)
117 | {
118 |       CLOSE(f->fd);
119 |       free(f);
120 | }
121 |
122 |
123 | /*
124 | **  xgetline -- get the next text line into memory
125 | **
126 | **  returns a pointer to the line (a NUL-terminated string) or a null pointer
127 | */
128 |
129 | char *xgetline(XFILE *f)
130 | {
131 |       char *s = f->nextChar, *p;
132 |       int n;
133 |
134 |       for (p = s; *p != 0; ++p)
135 |       {
136 |             if (*p == '\n')
137 |             {
138 |                   if (s < p && p[-1] == '\r')
139 |                         p[-1] = 0;
140 |                   else  *p = 0;
141 |                   f->nextChar = p + 1;
142 |                   return s;
143 |             }
144 |       }
145 |
146 |       /*
147 |       **  end of line not found in buffer -- p points to the sentinel NUL
148 |       */
149 |
150 |       if (p == f->buf)                    /* iff empty, EOF */
151 |             return NULL;
152 |
153 |       /*
154 |       **  move prefix of line to bottom of buffer
155 |       */
156 |
157 |       if (s != f->buf)
158 |       {
159 |             for (p = f->buf; (*p = *s) != 0; ++p, ++s)
160 |                   ;
161 |             s = f->buf;
162 |       }
163 |
164 |       n = XBUFSIZE - (p - f->buf);
165 |
166 |       if (n < QUANTUM)                    /* insufficient room, break line */
167 |       {
168 |             f->nextChar = p;
169 |             return s;
170 |       }
171 |
172 |       n = (n / QUANTUM) * QUANTUM;        /* quantize: count to read */
173 |       n = READ(f->fd, p, n);
174 |
175 |       /*
176 |       **  read error is sort of ignored here... same return as EOF.
177 |       **  we'll see if this proves to be sufficient...
178 |       */
179 |
180 |       if (n < 0)
181 |       {
182 |             f->nextChar = f->buf;
183 |             f->buf[0] = 0;
184 |             return NULL;
185 |       }
186 |
187 |       p[n] = 0;
188 |
189 |       for ( ; *p != 0; ++p)
190 |       {
191 |             if (*p == '\n')
192 |             {
193 |                   if (s < p && p[-1] == '\r')
194 |                         p[-1] = 0;
195 |                   else  *p = 0;
196 |                   ++p;
197 |                   break;
198 |             }
199 |       }
200 |
```

```
 201 |       f->nextChar = p;
 202 |       return p == s ? NULL : s;
 203 | }
```

TEXT STATISTICS

```
  5300 characters
   203 lines
```

LEXICAL STATISTICS

```
    12 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     4 constants [character]
     1 constants [string]
    26 constants [numeric]
```

SYMBOL TABLE

```
    CLOSE          104     118
    DOS_OPEN        47      93
    NULL            89     108     151     184     202
    O_RDONLY        93
    QUANTUM         70      71     166     172+
    READ            56      96     173
    XBUFN           66      67      71
    XBUFSIZE        71      86      96     164
    XFILE           84      86+     91     116     129
    __TURBOC__              46
    __ZTC__         46
    _dos_open       51
    _dos_read       60
    buf             56      60      91      96      99     100     150     157     159     161     164
           182     183
    count           58      60      61
    defined         46+
    error0          90     107
    error1          94     105
    error2          97     103
    f       86      89      91+     93      96+     99     100+    101     104     106     116     118
           119     129     131     141     150     157     159     161     164     168     173     182+
           183     201
    fcntl           43
    fd              56      60      93      96     104     118     173
    free           106     119
    h       41      42      43
    hdl             49      51      52
    len             56      60
    malloc          86
    mode            47      51
    n       87      96      99     132     164     166     172+    173+    180     187
    name            47      51      84      93
    nextChar       100     131     141     168     182     201
    p      131     134+    136     138+    139     140     141     150     159+    164     168     173
           187     189+    191     193+    194     195     196     201     202
    s      131     134     138     142     157     159+    161     169     193     202+
    size_t          56
    stdlib          41
    string          42
    xclose         116
    xgetline       129
    xopen           84
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  xfile.h -- definitions for fast line buffered files
  5 | */
  6 |
  7 | #ifndef __XFILE_H__
  8 | #define __XFILE_H__
  9 |
 10 | struct _xfile {
 11 |     int   fd;
 12 |     int   bufSize;
 13 |     char *buf;
 14 |     char *nextChar;
 15 |     char *lastChar;
 16 | };
 17 |
 18 | typedef struct _xfile XFILE;
 19 |
 20 | #include <dos.h>
 21 |
 22 | #if defined(__ZTC__)
 23 |  #include <io.h>
 24 |  #define DOS_OPEN dos_open
 25 |  #define READ     read
 26 |  #define CLOSE    close
 27 | #elif defined(__TURBOC__)
 28 |  #include <io.h>
 29 |  #include <fcntl.h>
 30 |  #define DOS_OPEN _open
 31 |  #define READ     _read
 32 |  #define CLOSE    _close
 33 | #else /* MSC */
 34 |  #include <stdlib.h>
 35 |  #include <fcntl.h>
 36 |  #define CLOSE    _dos_close
 37 | #endif
 38 |
 39 | XFILE *xopen(char const *);
 40 | void   xclose(XFILE *);
 41 | char  *xgetline(XFILE *);
 42 |
 43 | #endif  /* __XFILE_H__ */
```

TEXT STATISTICS

```
   792 characters
    43 lines
```

LEXICAL STATISTICS

```
     4 comments [std-C]
     0 comments [C++]
    20 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    CLOSE           26      32      36
    DOS_OPEN        24      30
    READ            25      31
    XFILE           18      39      40      41
    __TURBOC__              27
    __XFILE_H__              7       8
    __ZTC__         22
    _close          32
    _dos_close              36
    _open           30
    _read           31
    _xfile          10      18
    buf             13
    bufSize         12
    close           26
    defined         22      27
    dos             20
    dos_open        24
    fcntl           29      35
    fd              11
    h       20      23      28      29      34      35
    io              23      28
    lastChar        15
    nextChar        14
    read            25
    stdlib          34
    xclose          40
    xgetline        41
    xopen           39
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** XMS.C
**
** Routines to use Extended Memory from a DOS program.
** NOTE: Uses inline assembly language.
**
** Released to the public domain by Cliff Rhodes with no guarantees
** of any kind.
*/

#include <dos.h>

#include "xms.h"

#define XMS_INT  0x002f     /* DOS Multiplex interrupt */

static int (_far * XMSDriver)(void);

static int initFlag = 0;

/*
** Verify that an Extended Memory Manager is installed.
**
** Returns 1 if manager found, 0 if not.
**
** NOTE: This function should be called before any other XMS function!
*/

int XMSinit(void)
{
      union REGS regs;
      struct SREGS sregs;

      regs.x.ax = 0x4300;             /* Verify XMS manager present */
      int86(XMS_INT, &regs, &regs);
      if(regs.h.al == 0)
            return 0;

      regs.x.ax = 0x4310;             /* Get XMS manager entry point */
      int86x(XMS_INT, &regs, &regs, &sregs);

      /* Save entry point */

      XMSDriver = (int (_far *)(void)) MK_FP(sregs.es, regs.x.bx);

      return (initFlag = 1);
}

/*
** Return version number of XMS driver, or 0 if not available.
*/

int XMSversion(void)
{
      if(!initFlag)
            return 0;

      _asm mov ax, 0;

      return XMSDriver();
}

/*
** Returns number of bytes available in largest free block.
*/

long  XMScoreleft(void)
{
      if(!initFlag)
            return 0L;

      _asm mov ax, 0x0800;

      return 1024L * (long) XMSDriver();
}

/*
** Attempts to allocate size bytes of extended memory.
**
** Returns handle if successful, 0 if not.
**
** NOTE: Actual size allocated will be the smallest multiple of 1024
**       that is larger than size.
*/

unsigned int XMSalloc(long size)
{
      /* Get the number of 1024 byte units required by size. */

      int rval = (int) (size / 1024L);

      if(size % 1024L)
            rval++;    /* Add a block for any excess */

      if(!initFlag)
            return 0;

      _asm
```

```
101 |       {
102 |             mov dx, rval;
103 |             mov ax, 0x0900;
104 |       }
105 |
106 |       if(XMSDriver())
107 |             _asm mov rval, dx;
108 |       else  rval = 0;
109 |
110 |       return rval;
111 | }
112 |
113 | /*
114 | ** Attempts to free extended memory referred to by handle. Returns 1
115 | ** if successful, 0 if not.
116 | */
117 |
118 | int XMSfree(unsigned int handle)
119 | {
120 |       if(!initFlag)
121 |             return 0;
122 |
123 |       _asm
124 |       {
125 |             mov dx, handle;
126 |             mov ax, 0x0a00;
127 |       }
128 |
129 |       return XMSDriver();
130 | }
131 |
132 | typedef struct {
133 |    long         nbytes;     /* Number of bytes to move */
134 |    unsigned int shandle;    /* Handle of source memory */
135 |    long         soffset;    /* Offset of source in handle's memory area */
136 |    unsigned int dhandle;    /* Handle of destination memory */
137 |    long         doffset;    /* Offset of destination in memory */
138 | } XMSRequestBlock;
139 |
140 | static XMSRequestBlock bd;
141 |
142 | static long XMSMove(long n)
143 | {
144 |       long rval;
145 |       unsigned int segm, offs;
146 |       XMSRequestBlock _far *fptr = (XMSRequestBlock _far *) &bd;
147 |
148 |       if(!initFlag)
149 |             return 0L;
150 |
151 |       bd.nbytes = n;
152 |
153 |       offs = FP_OFF(fptr);
154 |       segm = FP_SEG(fptr);
155 |
156 |       _asm
157 |       {
158 |             push ds;      /* Save DS */
159 |             mov ds, segm;
160 |             mov si, offs;
161 |             mov ax, 0x0b00;
162 |       }
163 |
164 |       rval = (XMSDriver() == 0) ? 0L : n;
165 |
166 |       _asm pop ds;    /* Restore DS since we changed it to make this call */
167 |
168 |       return rval;
169 | }
170 |
171 | /*
172 | ** Attempts to copy n bytes from srchandle to desthandle memory areas.
173 | ** Returns number of bytes copied, or 0 on error.
174 | */
175 |
176 | long XMSmemcpy(unsigned int desthandle, long destoff,
177 |                unsigned int srchandle, long srcoff, long n)
178 | {
179 |       bd.shandle = srchandle;
180 |       bd.soffset = srcoff;
181 |       bd.dhandle = desthandle;
182 |       bd.doffset = destoff;
183 |
184 |       return XMSMove(n);
185 | }
186 |
187 | /*
188 | ** Attempts to copy n bytes from DOS src buffer to desthandle memory area.
189 | ** Returns number of bytes copied, or 0 on error.
190 | */
191 |
192 | int DOStoXMSmove(unsigned int desthandle, long destoff,
193 |                  const char *src, int n)
194 | {
195 |       bd.shandle = 0;
196 |       bd.soffset = (long) ((char _far *) src);
197 |       bd.dhandle = desthandle;
198 |       bd.doffset = destoff;
199 |
200 |       return (int) XMSMove((long) n);
```

```
 201 | }
 202 |
 203 | /*
 204 | ** Attempts to copy n bytes to DOS dest buffer from srchandle memory area.
 205 | ** Returns number of bytes copied, or 0 on error.
 206 | */
 207 |
 208 | int XMStoDOSmove(char *dest, unsigned int srchandle, long srcoff, int n)
 209 | {
 210 |       bd.shandle = srchandle;
 211 |       bd.soffset = srcoff;
 212 |       bd.dhandle = 0;
 213 |       bd.doffset = (long) ((char _far *) dest);
 214 |
 215 |       return (int) XMSMove((long) n);
 216 | }
```

TEXT STATISTICS

```
  4728 characters
   216 lines
```

LEXICAL STATISTICS

```
    23 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     1 constants [string]
    25 constants [numeric]
```

SYMBOL TABLE

```
    DOStoXMSmove            192
    FP_OFF          153
    FP_SEG          154
    MK_FP            46
    REGS             33
    SREGS            34
    XMSDriver        19      46      62      76     106     129     164
    XMSMove         142     184     200     215
    XMSRequestBlock         138     140     146+
    XMS_INT          17      37      42
    XMSalloc         88
    XMScoreleft              69
    XMSfree         118
    XMSinit          31
    XMSmemcpy       176
    XMStoDOSmove            208
    XMSversion               55
    _asm             60      74     100     107     123     156     166
    _far             19      46     146+    196     213
    al               38
    ax               36      41      60      74     103     126     161
    bd              140     146     151     179     180     181     182     195     196     197     198
          210       211     212     213
    bx               46
    dest            208     213
    desthandle              176     181     192     197
    destoff         176     182     192     198
    dhandle         136     181     197     212
    doffset         137     182     198     213
    dos              13
    ds              158     159     166
    dx              102     107     125
    es               46
    fptr            146     153     154
    h        13      38
    handle          118     125
    initFlag         21      48      57      71      97     120     148
    int86            37
    int86x           42
    mov              60      74     102     103     107     125     126     159     160     161
    n       142     151     164     177     184     193     200     208     215
    nbytes          133     151
    offs            145     153     160
    pop             166
    push            158
    regs             33      36      37+     38      41      42+     46
    rval             92      95     102     107     108     110     144     164     168
    segm            145     154     159
    shandle         134     179     195     210
    si              160
    size             88      92      94
    soffset         135     180     196     211
    src             193     196
    srchandle       177     179     208     210
    srcoff          177     180     208     211
    sregs            34      42      46
    x        36      41      46
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | ** XMS.H
  5 | **
  6 | ** Header file for Extended Memory routines in XMS.C.
  7 | */
  8 |
  9 | #ifndef _XMS_H
 10 | #define _XMS_H
 11 |
 12 | int  XMSinit(void);
 13 | int  XMSversion(void);
 14 | long XMScoreleft(void);
 15 | int  XMSfree(unsigned int handle);
 16 | long XMSmemcpy(unsigned int desthandle, long destoff,
 17 |                unsigned int srchandle, long srcoff, long n);
 18 | int  DOStoXMSmove(unsigned int desthandle, long destoff,
 19 |                const char *src, int n);
 20 | int  XMStoDOSmove(char *dest, unsigned int srchandle, long srcoff, int n);
 21 |
 22 | unsigned int XMSalloc(long size);
 23 |
 24 | #endif
```

TEXT STATISTICS

```
   606 characters
    24 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     3 preprocessor instructions
     0 constants [character]
     0 constants [string]
     0 constants [numeric]
```

SYMBOL TABLE

```
    DOStoXMSmove            18
    XMSalloc        22
    XMScoreleft             14
    XMSfree         15
    XMSinit         12
    XMSmemcpy       16
    XMStoDOSmove            20
    XMSversion              13
    _XMS_H           9      10
    dest            20
    desthandle              16      18
    destoff         16      18
    handle          15
    n       17      19      20
    size            22
    src             19
    srchandle       17      20
    srcoff          17      20
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
** XMSTEST.C
**
** Test extended memory routines in file XMS.C
**
** Released to the public domain by Cliff Rhodes with no guarantees
** of any kind.
*/

#include <stdio.h>
#include <stdlib.h>

#include "xms.h"

#define BufSize            30000
#define BlockSize  (long) 130000L

int main(void)
{
      int i;
      unsigned int handle1, handle2;       /* XMS handles */
      char *buf1, *buf2;

      if ((i = XMSinit()) == 0)
      {
            printf("Extended Memory Manager not found, terminating program\n");
            return 1;
      }
      i = XMSversion();
      printf("Extended Memory Manager Version %d.%d is installed\n",
             i >> 8, (i & 0xff));

      printf("There are %ld extended memory bytes available\n", XMScoreleft());

      buf1 = malloc(BufSize);
      buf2 = malloc(BufSize);
      if (buf1 == NULL || buf2 == NULL)
      {
            printf("Error allocating conventional memory\n");
            return 1;
      }
      for (i = 0; i < BufSize; i++)   /* Fill buf1 with some values */
      {
            buf1[i] = i % 255;
            buf2[i] = 0;
      }
      handle1 = XMSalloc(BlockSize);
      if (handle1 == 0)
      {
            printf("Error allocating XMS memory for handle 1\n");
            return 1;
      }
      printf("Handle 1 is %u and represents %ld bytes\n", handle1, BlockSize);
      printf("There are %ld extended memory bytes available\n", XMScoreleft());

      handle2 = XMSalloc(BlockSize);
      if (handle2 == 0)
      {
            XMSfree(handle1);
            printf("Error allocating XMS memory for handle 2\n");
            return 1;
      }
      printf("Handle 2 is %u and represents %ld bytes\n", handle2, BlockSize);
      printf("There are %ld extended memory bytes available\n", XMScoreleft());

      /* Copy data from DOS buf1 to XMS memory of handle1 */

      if (DOStoXMSmove(handle1, 0L, buf1, (long) BufSize) != BufSize)
          printf("DOStoXMS copy failed\n");

      /* Copy XMS handle1 data to XMS handle2 */

      if (XMSmemcpy(handle2, 0L, handle1, 0L, BlockSize) != BlockSize)
          printf("XMS copy failed\n");

      /* Copy XMS handle2 data to DOS buf2 */

      if (XMStoDOSmove(buf2, handle1, 0L, (long) BufSize) != BufSize)
          printf("XMStoDOS copy failed\n");

      /* buf1 data should now == buf2 data */

      for (i = 0; i < BufSize; i++)
      {
            if (buf1[i] != buf2[i])
            {
                  printf("*** ERROR: Mismatch in DOS buffers at "
                         "location %d ***\n", i);
                  break;
            }
      }
      free(buf1);
      free(buf2);

      XMSfree(handle2);
      printf("There are %ld bytes available after freeing handle 2\n",
             XMScoreleft());

}
```

```
 101 |       XMSfree(handle1);
 102 |       printf("There are %ld bytes available after freeing handle 1\n",
 103 |              XMScoreleft());
 104 |
 105 |       return 0;
 106 | }
```

TEXT STATISTICS

```
  3045 characters
   106 lines
```

LEXICAL STATISTICS

```
     8 comments [std-C]
     0 comments [C++]
     5 preprocessor instructions
     0 constants [character]
    18 constants [string]
    20 constants [numeric]
```

SYMBOL TABLE

```
    BlockSize       18      49      55      58      65      75+
    BufSize         17      37      38      44      70+     80+     85
    DOStoXMSmove            70
    NULL            39+
    XMSalloc        49      58
    XMScoreleft             35      56      66      99     103
    XMSfree         61      97     101
    XMSinit         26
    XMSmemcpy       75
    XMStoDOSmove            80
    XMSversion              31
    buf1            24      37      39      46      70      87      94
    buf2            24      38      39      47      80      87      95
    free            94      95
    h       12      13
    handle1         23      49      50      55      61      70      75      80     101
    handle2         23      58      59      65      75      97
    i       22      26      31      33+     44+     46+     47      85+     87+     90
    main            20
    malloc          37      38
    printf          28      32      35      41      52      55      56      62      65      66      71
            76      81      89      98     102
    stdio           12
    stdlib          13
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | /*
  4 | **  XSTRCAT.C - String concatenation function
  5 | **
  6 | **  Notes: 1st argument must be a buffer large enough to contain the
  7 | **          concatenated strings.
  8 | **
  9 | **          2nd thru nth arguments are the string to concatenate.
 10 | **
 11 | **          (n+1)th argument must be NULL to terminate the list.
 12 | */
 13 |
 14 | #include <stdarg.h>
 15 | #include "snip_str.h"
 16 |
 17 | #if defined(__cplusplus) && __cplusplus
 18 |  extern "C" {
 19 | #endif
 20 |
 21 | char *xstrcat(char *des, char *src, ...)
 22 | {
 23 |         char *destination = des;
 24 |         va_list v;
 25 |
 26 |         va_start(v, src);
 27 |
 28 |         while (src != 0)
 29 |         {
 30 |                 while (*src != 0)
 31 |                         *des++ = *src++;
 32 |                 src = va_arg(v, char *);
 33 |         }
 34 |         *des = 0;
 35 |
 36 |         va_end(v);
 37 |
 38 |         return destination;
 39 | }
 40 |
 41 | #if defined(__cplusplus) && __cplusplus
 42 |  }
 43 | #endif
```

TEXT STATISTICS

```
   887 characters
    43 lines
```

LEXICAL STATISTICS

```
     2 comments [std-C]
     0 comments [C++]
     6 preprocessor instructions
     0 constants [character]
     2 constants [string]
     3 constants [numeric]
```

SYMBOL TABLE

```
    __cplusplus             17+     41+
    defined         17      41
    des             21      23      31      34
    destination             23      38
    h       14
    src             21      26      28      30      31      32
    stdarg          14
    v       24      26      32      36
    va_arg          32
    va_end          36
    va_list         24
    va_start        26
    xstrcat         21
```

```
/* +++Date last modified: 05-Jul-1997 */

/*
**  XSTRCMP.C - Simple string pattern matching functions using DOS
**              wildcards ('?' & '*').
**
**  Derived from code by Arjan Kentner (submitted by Steve Summit),
**  modified by Bob Stout.
*/

#include <stdio.h>                  /* For NULL                          */
#include <ctype.h>                  /* For toupper()                     */
#include <assert.h>                 /* For assert()                      */
#include "sniptype.h"               /* For Boolean_T                     */
#include "dirent.h"                 /* Validate prototypes, also
                                       includes extkword.h for PASCAL   */

/*
**  Arguments: 1 - Pattern to match
**             2 - String to match
**
**  Returns: True_ if match
**           False_ if no match
**           Error_ if passed a null pointer (see below)
**
**  Notes: 1. Two versions are supplied, one case-sensitive and one not.
**         2. Each version consists of a recursive static function which
**            does all the work, and a wrapper which checks for null
**            pointers before calling the static function.
**         3. If assert() is enabled (i.e. if NDEBUG is undefined or false),
**            the wrapper functions will abort with an assertion error in
**            case a null pointer is passed.
**         4. If assert() is disabled (i.e. if NDEBUG is defined), the
**            wrapper function will return Error_ to the calling program in
**            case a null pointer is passed.
*/

/*
**  Case-sensitive version
*/

static Boolean_T PASCAL patmat (const char *pat, const char *str)
{
      switch (*pat)
      {
      case '\0':
            return !*str;

      case '*' :
            return patmat(pat+1, str) || *str && patmat(pat, str+1);

      case '?' :
            return *str && patmat(pat+1, str+1);

      default  :
            return (*pat == *str) && patmat(pat+1, str+1);
      }
}

Boolean_T xstrcmp (const char *pat, const char *str)
{
      assert(str && pat);
      if (NULL == str || NULL == pat)
            return Error_;
      else  return(patmat(pat, str));
}

/*
**  Case-insensitive version
*/

static Boolean_T PASCAL patimat (const char *pat, const char *str)
{
      switch (*pat)
      {
      case '\0':
            return !*str;

      case '*' :
            return patimat(pat+1, str) || *str && patimat(pat, str+1);

      case '?' :
            return *str && patimat(pat+1, str+1);

      default  :
            return (toupper(*pat) == toupper(*str)) && patimat(pat+1, str+1);
      }
}

Boolean_T xstricmp (const char *pat, const char *str)
{
      assert(str && pat);
      if (NULL == str || NULL == pat)
            return Error_;
      else  return(patimat(pat, str));
}

#ifdef TEST

#include <stdio.h>
```

```
 101 |
 102 | main(int argc, char *argv[])
 103 | {
 104 |       if (3 != argc)
 105 |       {
 106 |             puts("Usage: XSTRCMP mask string");
 107 |             return -1;
 108 |       }
 109 |       printf("xstrcmp(\"%s\", \"%s\") returned %d\n", argv[1], argv[2],
 110 |             xstrcmp(argv[1], argv[2]));
 111 |
 112 |       printf("xstricmp(\"%s\", \"%s\") returned %d\n", argv[1], argv[2],
 113 |             xstricmp(argv[1], argv[2]));
 114 |
 115 |       printf("xstricmp(NULL, \"%s\") returned %d\n", argv[2],
 116 |             xstricmp(NULL, argv[2]));
 117 |
 118 |       printf("xstricmp(\"%s\", NULL) returned %d\n", argv[1],
 119 |             xstricmp(argv[1], NULL));
 120 |
 121 |       printf("xstricmp(NULL, NULL) returned %d\n", xstricmp(NULL, NULL));
 122 |
 123 |       return 0;
 124 | }
 125 |
 126 | #endif /* TEST */
```

TEXT STATISTICS

```
  3652 characters
   126 lines
```

LEXICAL STATISTICS

```
    11 comments [std-C]
     0 comments [C++]
     8 preprocessor instructions
     6 constants [character]
     8 constants [string]
    27 constants [numeric]
```

SYMBOL TABLE

```
    Boolean_T       42      60      72      90
    Error_          64      94
    NULL            63+     93+    116     119     121+
    PASCAL          42      72
    TEST            98
    argc           102     104
    argv           102     109+    110+    112+    113+    115     116     118     119
    assert          13      62      92
    ctype           12
    h       11      12      13     100
    main           102
    pat             42      44      50+     53      56+     60      62      63      65      72      74
            80+     83      86+     90      92      93      95
    patimat         72      80+     83      86      95
    patmat          42      50+     53      56      65
    printf         109     112     115     118     121
    puts           106
    stdio           11     100
    str             42      47      50+     53+     56+     60      62      63      65      72      77
            80+     83+     86+     90      92      93      95
    toupper         86+
    xstrcmp         60     110
    xstricmp        90     113     116     119     121
```

```
  1 | /* +++Date last modified: 05-Jul-1997 */
  2 |
  3 | #include <stdio.h>
  4 | #include "xfile.h"
  5 |
  6 | #ifdef __WATCOMC__
  7 |  #pragma off (unreferenced);
  8 | #endif
  9 | #ifdef __TURBOC__
 10 |  #pragma argsused
 11 | #endif
 12 |
 13 | int main(int argc, char **argv)
 14 | {
 15 |       while (*++argv != 0)
 16 |       {
 17 |             XFILE *f = xopen(*argv);
 18 |
 19 |             if (f == 0)
 20 |                   fprintf(stderr, "ERROR: can't open file %s\n", *argv);
 21 |             else
 22 |             {
 23 | #if 0
 24 |                   char *s;
 25 |
 26 |                   fprintf(stdout, "--- %s ---\n", *argv);
 27 |                   while ((s = xgetline(f)) != NULL)
 28 |                         fputs(s, stdout);
 29 |                   xclose(f);
 30 | #else
 31 |                   unsigned int nLines = 0;
 32 |                   char *s;
 33 |
 34 |                   while (xgetline(f) != NULL)
 35 |                         ++nLines;
 36 |                   printf("%5u lines in %s\n", nLines, *argv);
 37 |                   xclose(f);
 38 | #endif
 39 |             }
 40 |       }
 41 |
 42 |       return 0;
 43 | }
```

TEXT STATISTICS

```
   969 characters
    43 lines
```

LEXICAL STATISTICS

```
     1 comments [std-C]
     0 comments [C++]
    11 preprocessor instructions
     0 constants [character]
     4 constants [string]
     5 constants [numeric]
```

SYMBOL TABLE

```
    NULL            27      34
    XFILE           17
    __TURBOC__               9
    __WATCOMC__              6
    argc            13
    argsused        10
    argv            13      15      17      20      26      36
    f       17      19      27      29      34      37
    fprintf         20      26
    fputs           28
    h       3
    main            13
    nLines          31      35      36
    off              7
    printf          36
    s       24      27      28      32
    stderr          20
    stdio            3
    stdout          26      28
    unreferenced             7
    xclose          29      37
    xgetline        27      34
    xopen           17
```