Ecriture itérative ou récursive

Exemple 1 : les suites récurrentes d'ordre 2

Objectif : calculer les valeurs d'une suite définie par ses premiers termes et une relation de récurrence

Données :
Les premiers termes *a*, *b*.
La fonction de deux variables f telle que $u_n=f(u_{n-1},u_{n-2})$
L'indice *n* du terme cherché

Description de l'algorithme itératif

Il est nécessaire, à chaque itération, de stocker le résultat (val1 := u_{k-1}) et le résultat précédent (val0 := u_{k-2}). Après l'itération, il faut que val1 contienne u_k et val0 contienne u_{k-1}. Les deux affectations paraissent incompatibles :
val1 := f(val1,val0) qui écrase la valeur u_{k-1} ;
val0 := val1, qui perd u_{k-2}
On résout la question en stockant val1 dans une variable temporaire temp.

Implémentation MAPLE

```
> suite:=proc(a,b,n)
local val0,val1,temp,resultat,k;
val0:=a; val1:=b;
for k to n do #boucle si n>1
     temp:=val1;
     val1:=f(val1,val0);
     val0:=temp
end do;
if n=0 then resultat:=val0 else resultat:=val1 end if;
resultat #valeur finale
end proc;
```

```
suite := proc(a, b, n)
local val0, val1, temp, resultat, k;
    val0 := a;
    val1 := b;
    for k to n do temp := val1; val1 := f(val1, val0); val0 := temp end do;
    if n = 0 then resultat := val0 else resultat := val1 end if;
    resultat
end proc
```

Description de l'algorithme récursif

u_n est défini en fonction de ses prédécesseurs. On n'oublie pas de définir les premiers termes.

Implémentation MAPLE

```
> u := proc(n) global f ;
if n=0 then a # définition de u0
elif n=1 then b # définition de u1
else f(u(n-1),u(n-2)) # appel récursif
end if
end proc ;
```

```
u := proc(n)
global f;
    if n = 0 then a elif n = 1 then b else f(u(n − 1), u(n − 2)) end if
end proc
```

NB : pour que cette procédure soit opérationnelle, il faut bien sûr définir la procédure *f*, par ex :

```
f := (x,y)-> x+y ; # pour la suite de Fibonacci
```

Avantages et inconvénients

L'écriture récursive est plus simple, l'algorithme est clair, facile à comprendre pour le lecteur (donc à privilégier dans une épreuve d'algorithmique, par exemple).

Cependant, elle est gourmande pour la mémoire car elle nécessite de calculer et stocker simultanément un grand nombre de valeurs intermédiaires.

L'écriture itérative, elle, n'utilise que trois variables val0,val1,temp. Elle est souvent moins lisible, mais plus économique pour la mémoire. Elle est donc à privilégier dans la pratique, dans les cas où la machine dont on dispose a une capacité mémoire limitée ou si n est très grand (NB : les besoins d'un algorithme récursif en terme de mémoire sont souvent de type exponentiel, et on dépasse vite les capacités d'un ordinateur même très puissant).

Une « astuce » : l'option remember. La procédure récursive calcule de nombreuses valeurs intermédiaires plusieurs fois. Par exemple, pour calculer u(20), la procédure récursive calcule u(19) et u(18). Pour calculer u(19), elle va calculer u(18) et u(17). Si on ne fait rien, u(18) sera donc calculé 2 fois, u(17) 3 fois, et u(2)… 4181 fois ! (En fait, pour calculer u(n), on doit calculer u(p) F_{n-p} fois, où F_k est le terme de la suite de Fibonacci - donc d'ordre exponentiel.) L'option *remember* permet à une procédure de stocker en mémoire toute valeur déjà calculée. On peut ainsi faire retomber le temps de calcul dans des limites raisonnables :

```
u := proc(n) option remember ; [...] end proc ;
```

NB : l'option *remember* ne doit pas être utilisée pour l'épreuve d'algorithmique