Structures itératives (boucles)

1. Qu'est-ce qu'une boucle ?

Une **boucle** (en anglais, *loop*) est une structure algorithmique qui demande de **répéter** l'exécution d'une ou plusieurs instructions (on parle de "structure itérative"). Chaque « tour de boucle » est une **itération**.

En Maple, les instructions à répéter sont encadrées par les mots clés **do** et **end do** (ou **od**)

Exemple :
> **do** instruction1 ; instruction2 ; … **end do** ;

Dans l'exemple, l'exécution de cette boucle ne s'arrête jamais (=*plantage* du logiciel).
Dans la pratique, une boucle doit toujours être assortie d'une **condition de sortie** qui assure que son exécution s'arrêtera après un temps fini.
On distingue trois structures possibles.

2. Structure « répéter [instructions] jusqu'à [condition de sortie] »

Il suffit d'insérer une instruction conditionnelle de sortie (**break**) dans la boucle.

Exemple
> n := 0 ;
> **do** n :=n+3 ; **if** n>10 **then** break **end if end do** ;
> n ;

IMPORTANT Dans une telle structure, les instructions de la boucle écrites *avant* l'instruction de sortie sont exécutées *au moins une fois* (on exécute, *puis* on se demande s'il faut répéter).

Cas des procédures :
A l'intérieur d'une procédure, les commandes *return* et *error* permettent également de sortir de la boucle. L'exécution de la procédure tout entière est alors stoppée.
>exemple1 := **proc**()
n := 0 ;
do n :=n+3 ; **if** n>10 **then** return(1000) **end if end do** ;
end proc ;
> **exemple1();**

1000

>exemple 2 := **proc**()
n :=0 ;
do n :=n+3 ; **if** n>10 **then** error(`message d'erreur`) **end if end do** ;
end proc ;
> **exemple2();**
Error, (in exemple2) message d'erreur

NB : *return* et *error* sont des commandes de terminaison de procédure (cf. fiche « procédures »). L'évaluation de la boucle s'arrête parce que celle de la procédure s'arrête.

3. Structure « tant que [condition d'entrée] exécuter [instructions] » : la boucle "while"

while ("tant que") : la boucle s'éxécute "tant que" la condition qui suit **while** prend la valeur **true**, et s'arrête dès que cette condition prend la valeur **false** ou ***FAIL***.

Exemple
> x:=0; # initialisation (sinon, la condition x <= 10 n'a pas de sens)
> **while** x<=10 **do** x:=x+2 **end do** :
> x ;

12

La boucle s'arrête quand la condition n'est plus vraie.

Condition d'entrée
C'est une expression booléenne (comme x>1) qui doit être évaluable lors de l'exécution. Il faut donc veiller à :

1) Initialiser les variables (**while** x>1 **do** … **od** ne sera pas exécutée si x n'a pas de valeur) ;

2) Les comparaisons doivent se faire soit entre variables de type *numeric* (entiers, *floats*), soit entre variables de type string. Par exemple, il ne faut pas écrire :

> **while** k<sqrt(2) **do** … **od**;

Mais :

> **while** k<evalf(sqrt(2)) **do** … **od** ;

IMPORTANT Dans une boucle « tant que », si la condition d'entrée est fausse, les instructions de la boucle *ne seront pas* exécutées, même une seule fois (on se demande s'il faut exécuter les instructions *avant* de commencer).

ATTENTION Ne pas confondre avec la structure conditionnelle !
> while (x<=10) then instruction # ???
> if x<=10 then do x:=x+2 od fi; # ??? mauvaise structure de contrôle

4. Structure itérative incrémentielle : la boucle"for"

L'évaluation d'une boucle crée un « **compteur** », c'est-à-dire une variable (*interne* et locale) qui prend la valeur initiale 1 et qui est *incrémenté* (=augmenté) de 1 après chaque itération.
La valeur du compteur peut être récupérée et stockée dans une variable (globale) définie par le mot-clé **for**.

to
Le contrôle "**to**" (*jusqu'à*) limite le nombre d'itérations de la boucle. La boucle s'arrête lorsque la valeur du compteur est strictement supérieure à celle indiquée par **to** :
> **to** 2 **do** "hello" **end do**;
Il est à noter que le compteur est incrémenté *après* chaque itération, même la dernière, c'est-à-dire que dans l'exemple, le mot « hello » sera écrit 2 fois, la valeur finale du compteur étant 3 (= 1 + nombre d'itérations).

Explicitons les actions de l'interpréteur Maple :

initialisation compteur :=1
test compteur>2 ? faux, donc "hello";
incrémentation compteur :=compteur+1 (donc compteur = 2)
test compteur>2 ? faux, donc "hello";
incrémentation compteur :=compteur+1 (donc compteur = 3)
test compteur >2 ? vrai donc fin de boucle

NB le test de sortie est du type > et non =, on peut donc indiquer une valeur finale non entière (mais *numeric* tout de même !)

from
Le contrôle "from" (*à partir de*) permet d'initialiser le compteur avec une valeur (*numeric)* au choix
> **from** 1.9 **to** 3.9 **do** "hello" **end do**;
On peut également utiliser des valeurs de type caractère (*character*)
> **from** "a" **to** "z" **do** instruction **end do** ;

by
Par défaut, le compteur est incrémenté de 1 à chaque itération (c'est-à-dire, par défaut, 1, 2, 3, etc.) Le contrôle "by" permet de changer le *pas* de l'incrémentation.
> **by** 2 **to** 10 **do** "hello" **end do**;
Dans cet exemple, le compteur prendra les valeurs 1, 3, 5, 7, 9, 11.

for
"for k" définit une variable (globale) k, qui est affectée de la valeur du compteur.
On l'utilise souvent en combinaison avec les contrôles "to", "from" et "by"
Exemple
> **for** k **from** 0 **to** 3 **by** 0.6 **do** *k* **od**;

Ordre des contrôles
1) Aucun de ces contrôles n'est indispensable pour la correction de la syntaxe.

2) **for**, s'il est présent, **doit être écrit en premier** ; l'ordre de **from, to, by** est indifférent.

Décrémentation
Il est possible d'utiliser un compteur « décrémentiel » en imposant un pas négatif.

```
> for k from 5 to 3 by -1 do k od;
```

5

4

3

```
> k;
```

2

in
Il est possible d'indexer les itérations à l'aide de valeurs choisies par l'utilisateur :

```
> S:=7,sqrt(2), b:
for k in S do k od;
```

$$7$$
$$\sqrt{2}$$
$$b$$

k prend successivement les valeurs des termes de S.
On peut utiliser une liste ou un ensemble plutôt qu'une séquence.

5. Application : définition d'une séquence
On cherche à définir la séquence (u(0),…,u(10)) des premiers termes d'une suite définie par récurrence :

$$u(n+1)=f(u(n)) \ ;$$

On commence par initialiser les variables :

```
> u := 0 :        # initialisation de u (ici, on suppose u(0)=0).
> S := u :        # initialisation de S
```

On remplit la séquence à l'aide d'une boucle :

```
> for k to 10 do u :=f(u) ; S :=S,u od:
```

S prend successivement les valeurs 0, puis 0,f(0), etc.

NB : Dans d'autres cas, on peut initialiser avec la séquence « vide », (mot-clé *NULL)*

```
> S := NULL :     # initialisation par une séquence vide
```