Les boucles

I. La boucle "Tant Que"

La structure est la suivante :

Tant que (condition)
action . . .
Fin Tant que

Lorsque l'instruction "Tant Que" est rencontrée, la condition est évaluée. Si elle est réalisée, on exécute l'action spécifiée et lorsque le "Fin Tant Que" est rencontré on revient au "Tant Que" et on recommence. Si elle n'est pas réalisée, on ignore l'action et on passe directement à l'instruction suivant le "Fin Tant Que" (sortie de boucle).
Voici par exemple une séquence qui affiche l'alphabet en majuscule :

```
. . .
n ← 65
Tant que (n ⩽ 90)
    afficher(carASCII(n))
    n ← n+1
Fin Tant Que
. . .
```

On notera l'importance de l'initialisation des variables utilisées dans la condition : si n était initialisé par 95, les instructions situées dans la boucle ne seraient jamais exécutées. Par ailleurs, les variables utilisées dans la condition doivent évoluer dans le contenu de la boucle . . . faute de quoi on ne pourrait plus en sortir une fois rentré ! (enlevez le n ← n + 1 dans l'exemple précédent et vous verrez).
Et une mauvaise affectation peut avoir le même effet. Le seul moyen de ne pas tomber dans le piège des boucles infinies (ou jamais parcourues !) est de tester son algorithme à la main sur des cas simples . . . et là les failles apparaissent. Essayez sur le morceau d'algorithme suivant :

```
. . .
n ← 5
Tant que n ≠ 10
    afficher(n)
    n ← n+2
Fin Tant Que
. . .
```

Vous remarquerez dans ce cas que le problème peut disparaitre de différentes façons :
. par exemple en remplaçant n ← n + 2 par n ← n + 1
. ou bien en remplaçant n← 5 par n← 4
. ou encore en remplaçant la condition n ≠10 par n < 10

II. La boucle "Pour"

Il s'agit d'une boucle "comptée" dans laquelle le programme s'occupe de tout : initialisation de la variable de comptage, son évolution et la condition d'arrêt. Sa structure est la suivante :

Pour VariableComptage **allant de ValeurDébut à ValeurFin**
    action . . .
Fin pour
instruction suivante

Lorsque l'instruction "Pour" est rencontrée la première fois, le protocole suivant s'enclenche :

1. affecter VariableComptage par ValeurDebut
2. tester la condition (VariableComptage <ValeurFin) :
    Si elle est vraie aller au point 3
    Si elle est fausse aller au point 6
3. exécuter la séquence d'instructions de l'action
4. une fois en "Fin pour", incrémenter VariableComptage (augmenter de 1)
5. aller au point 2
6. continuer l'algorithme à l'instruction "instruction suivante"

On remarquera que cette boucle est équivalente à la structure "Tant Que" suivante :

```
VariableComptage ← ValeurDebut
Tant que (VariableComptage ⩽ ValeurFin)
   action . . .
   VariableComptage ← VariableComptage + 1
Fin Tant que
instruction suivante
```

Quatre points extrêmement importants sont à noter :

- l'action n'est jamais réalisée si ValeurDebut > ValeurFin
- si ValeurDebut ≤ValeurFin, l'action sera exécutée exactement (ValeurFin – ValeurDebut +1) fois
- dans l'action, le contenu de VariableComptage ne doit en aucun cas être modifié
- à la sortie de la boucle "pour", VariableComptage contient la valeur (ValeurFin + 1)

L'exemple du paragraphe précédent aurait pu aussi s'écrire comme ceci :

```
. . .
Pour n allant de 65 à 90
   afficher(carASCII(n))
n suivant
. . .
```

III. Imbrication

Là encore, les structures peuvent être imbriquées les unes dans les autres (boucle dans une boucle).
Par exemple, la séquence suivante affiche les tables de multiplications de 2 à 9 :

```
. . .
Pour i allant de 2 à 9
   afficher("Table des ",i," :")
   Pour j allant de 1 à 10
      afficher(i , " fois " , j , ' = ' , i×j)
   j suivant
i suivant
. . .
```