CHAPITRE4

LES STRUCTURES ITÉRATIVES

1.Introduction :

Souvent dans un problème informatique on peut rencontrer des cas ou le même traitement s'exécute de la même façon un certain nombre de fois, le nombre de fois d'exécution des instructions peut être connu à l'avance ou non ; par exemple si on veut afficher les 100 premiers entiers commençant par 1 on devrait procéder ainsi :

```
Algorithme Affichage
Début
Ecrire(1)
Ecrire(2)
Ecrire(3)
…
Ecrire(100)
Fin
```

On remarque que la même instruction Ecrire () se répète 100 fois ce qui est pénible à faire pour un programmeur et si on changeait 100 par 1000 ou par 10000 là ça devient de plus en plus compliquer et l'algorithme aura des milliers de lignes d'instructions; dans ces cas la les structures itératives offrent une solution très élégante à ce genre de situation.

On peut donc dire que les structures itératives sont des notions fondamentales de l'algorithmique et dont le rôle est de répète un même traitement autant de fois que l'on veut pour vue que la condition d'exécution soit satisfaite. Cette structure s'arrête lorsque cette condition n'est plus vérifiée. Il existe trois schémas de représentation d'une structure répétitive à savoir :

- le schéma pour

- le schéma Tant que
- le schéma Répéter… jusqu'à

L'utilisation d'un tel ou tel schéma dépend essentiellement de la nature du problème à résoudre.

2. La structure Pour :

Généralement la boucle « Pour » n'est utilisée que si on connait à l'avance le nombre d'itérations ; cette structure itère le même traitement pour une plage de valeurs entières comprises entre une borne inférieure et une borne supérieure. La mise à jour étant automatique, l'arrêt du traitement de la boucle Pour se réalise lorsqu'on dépasse l'une des bornes.

Syntaxe

Pour compteur **De** val_init **à** val_final,[Pas=valeur] **Faire**

Bloc d'instructions

Fin pour

L'organigramme ci dessous illustre le fonctionnement de la structure Pour

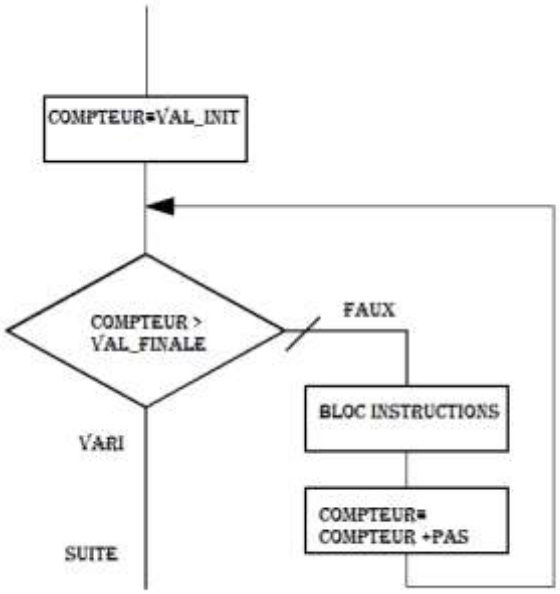

Figure 8- L'organigramme de l'exécution le la boucle « Pour »

Initialement le compteur prend la valeur initiale ; avant l'exécution du bloc d'instructions un test est réalisé pour savoir si le compteur a dépassé la valeur finale ou non si ce n'est pas le cas le bloc d'instruction est exécuté et la valeur du compteur sera incrémentée de façon automatique par la valeur Pas ; dans le cas contraire on sort de la boucle et le reste de l'algorithme est exécuté.

Remarques :

- une boucle Pour peut être exécutée 0,1 ou n fois.
- On utilise la boucle Pour que si on connaisse le nombre d'itération à l'avance.
- Le Pas d'incrémentation de la boucle peut être négatif → sens de la boucle est décroissant ou positif → sens de la boucle est croissant.
- Si le Pas n'est pas indiquer le Pas par défaut sera 1 ou -1.
- Si la valeur du Pas = 0, on sera en présence d'une boucle infinie.

Exercice d'application 1 :

Réécrire l'algorithme élaboré précédemment en utilisant la boucle pour.

Correction :

```
Algorithme Affichage
  Variable
   I : entier
    Début
      Pour I de 1 à 100 Faire
        Ecrire(I)
      Fin Pour
    Fin
```

Exercice d'application 2 :

Refaire le même exercice mais dans ce cas on va afficher les N premiers entiers commençant par 1 et ou N est saisie au clavier par l'utilisateur.

Correction :

```
Algorithme Affichage
    Variable
     I, N : entier
     Début
       Ecrire("donner un entier")
       Lire(N)
         Pour I de 1 à N Faire
             Ecrire(I)
        Fin Pour
     Fin
```

Trace de l'exécution :

1

2

…

N

Exercice d'application 3 :

Refaire le même exercice mais cette fois le sens de la boucle sera négatif

Correction :

```
Algorithme Affichage
   Variable
   I, N : entier
    Début
      Ecrire("donner un entier")
      Lire(N)
        Pour I de N à 1 Faire
          Ecrire(I)
        Fin Pour
    Fin
```

Trace de l'exécution :

N

N-1

…

1

Exercice d'application 4 :

Ecrire un algorithme qui permet de saisir un entier N et d'afficher tous les nombres impairs compris entre 1 et N

Correction :

```
Algorithme Impaire
    Variable
    i, N : entier
     Début
       Ecrire("donner un entier")
       Lire(N)
         Pour i de 1 à N Faire
            Si (i Mod 2 # 0) alors
              Ecrire(i)
           Fin si
         Fin Pour
      Fin
```

Trace d'exécution :

N=10

1 3 5 7 9

3. La structure « Tant que …Faire » :

La boucle tant que permet d'exécuter un ensemble d'instructions tant que la condition est vérifiée dès que cette condition n'est plus vérifiée on sort de la boucle.

Syntaxe

```
Tant que (condition vérifiée) Faire
Bloc d'instruction
Fin Tant que
```

L'organigramme ci dessous illustre le fonctionnement de cette structure :

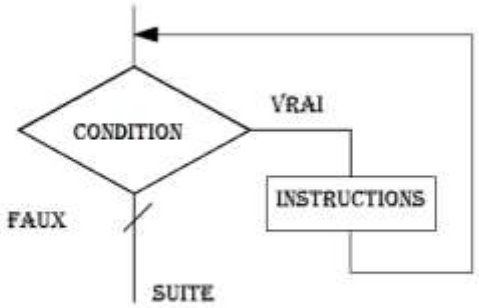

Figure 9- L'organigramme de l'exécution le la boucle «Tant que…Faire »

Dans cette structure on commence par tester la condition si elle est vérifiée l'ensemble d'instructions est exécuté.

Remarques :

- A la fin de chaque itération de la boucle Tant que Il est indispensable d'initialiser correctement les variables de la condition d'exécution et de les mettre à jour : condition nécessaire et obligatoire pour pouvoir reboucler.
- La mise à jour de ces variables peut se faire soit par une lecture, soit par une affectation.
- La structure Tant que est conseillée surtout pour les problèmes indiquant dans leur énoncé une condition d'exécution.
- Une condition d'exécution est la négation de la condition d'arrêt.
- La boucle Tant que peut être exécutée 0, 1 ou n fois.

Exercice d'application 5 :

Réécrire l'exercice 4 en utilisant la boucle Tant que.

Correction :

```
Algorithme Impaire
    Variable
     i, N : entier
     Début
      Ecrire("donner un entier")
      Lire(N)
      i← 1
        Tant que (i <= N) Faire
          Si (i Mod 2 # 0) alors
            Ecrire(i)
          Fin si
          i←i+ 1
        Fin Tant que
    Fin
```

4. La structure « Répéter … Jusqu'à » :

La boucle Répéter permet de rentrer dans la boucle quelque soit la condition et réitère l'exécution jusqu'à ce que la condition soit vérifiée.

Syntaxe :

Répéter

Bloc d'instructions

Jusqu'à (condition soit vrai)

L'organigramme ci-dessous illustre le fonctionnement de cette structure.

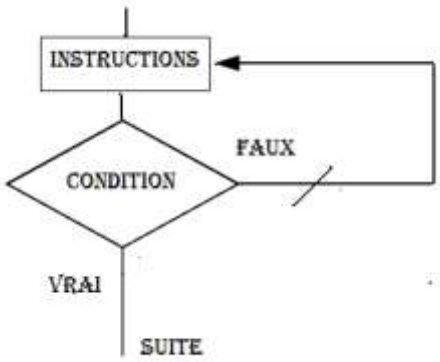

Figure 10- L'organigramme de l'exécution le la boucle «Répéter…Jusqu'à »

Dans cette structure le traitement est exécuté une première fois puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée.

Remarques :

- Dans une boucle Répéter, le traitement est exécuté au moins une seule fois quelle que soit la valeur de la condition d'arrêt.
- Dans une boucle Répéter, on parle de condition d'arrêt ; quant elle est atteinte, on sort de cette boucle.
- Il est indispensable d'initialiser correctement les variables de la condition d'arrêt et de les mettre à jour à la fin de chaque itération : condition nécessaire et obligatoire pour pouvoir reboucler.
- La structure Répéter est conseillée surtout pour les problèmes indiquant dans leur énoncé une condition d'arrêt.

Exercice d'application 6 :

Ecrire un algorithme qui permet de saisir un entier strictement positif et refusé toute valeur inférieur ou égal à zéro en utilisant la boucle Répéter … Jusqu'à.

Correction :

```
Algorithme Saisie
    Variable
    N : entier
  Début
      Répéter
         Ecrire("donner un entier")
         Lire(N)
      Jusqu'à ( N > 0)
   Fin
```

Exercice d'application 7 :

Ecrire un algorithme qui permet de saisir un entier N puis de calculer son factoriel

Rappelle factoriel de N =1*2*3*4*…*N ; faites 3 versions à chaque fois utilisez une boucle différente.

1ère version

```
Algorithme Factoriel 1
    Variables
    i, f, N : entier
  Début
     Ecrire("donner un entier")
     Lire (N)
    f←1
       Pour  i de 2 à N faire
        f←f*i
```

```
        Fin pour
        Ecrire("N != ",f)
    Fin
```

2ème version

```
Algorithme Factoriel 2
    Variables
    i, f, N : entier
  Début
    Ecrire("donner un entier")
    Lire (N)
    f←1
    i←2
      Tant que (i <= N) Faire
        f←f*i
        i←i+1
      Fin Tant que
   Ecrire("N != ",f)
  Fin
```

3ème version

```
Algorithme Factoriel 3
    Variables
    i, f, N : entier
    Début
        Ecrire("donner un entier")
        Lire (N)
        f ←1
        i←1
          Répéter
          f←f*i
          i←i +1
        Jusqu'à (i > N)
    Ecrire("N != ",f)
Fin
```

5. Choix de la structure itérative :

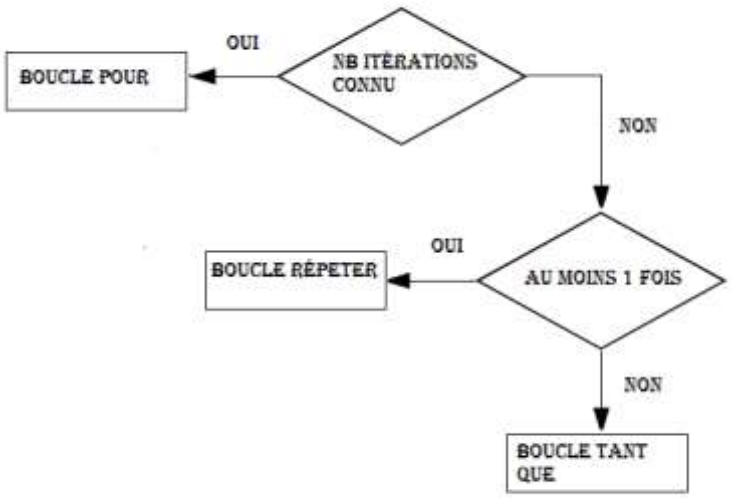

Figure 11- choix de la structure itérative

D’après la figure, si on a un nombre connu d’itération on opte pour la boucle « Pour » sinon si on est sur que le traitement s’exécute au moins une fois on opte pour la boucle « Répéter … Jusqu’à » sinon on choisi la boucle « Tant que ».