LES STRUCTURES REPETITIVES

1. Introduction

Dans les problèmes quotidiens, on ne traite pas uniquement des séquences d'actions, sous ou sans conditions, mais il peut être fréquent d'être obligé d'exécuter un traitement (séquence d'actions) plusieurs fois. En effet, pour saisir les N notes d'un étudiant et calculer sa moyenne, on est amené à saisir N variables, puis faire la somme et ensuite diviser la somme par N. Cette solution nécessite la réservation de l'espace par la déclaration des variables, et une série de séquences d'écriture/lecture. Ce problème est résolu à l'aide des structures répétitives. Celles ci permettent de donner un ordre de répétition d'une action ou d'une séquence d'actions une ou plusieurs fois.

2. Les structures répétitives

2.1. La boucle POUR

Cette structure exprime la répétition d'un traitement un nombre de fois.

Syntaxe

POUR ***compteur*** **DE** ***Valeur_initial*** **A** ***Valeur_finale*** **[PAS** ***Valeur_pas*****] FAIRE**
{Traitement}
FINFAIRE

Où ***compteur*** est une variable entière, qui compte le nombre de répétition du <Traitement>,
Valeur_initiale : la valeur initiale à laquelle le *compteur* est initialisé,
Valeur_finale : la valeur finale à laquelle se termine le *compteur*,
Valeur_pas : la valeur du pas, c'est la valeur qu'on rajoute au *compteur* à chaque fin de traitement.

Remarques

1. La boucle POUR est utilisée lorsqu'on connaît le nombre de répétition du <Traitement> **d'avance**.
2. La valeur du pas peut être positive ou négative et par conséquent, il faut, au départ de la boucle, que la *Valeur_initiale* <= la *Valeur_finale* ou la *Valeur_initial* e>= la *Valeur_finale* selon la positivité ou la négativité de cette valeur.
3. La valeur du pas est égale à 1 par défaut.

2.1.1. Les étapes d'exécution de la boucle POUR

1. Initialisation de Compteur par la valeur de *Valeur_initiale* (comme si on avait *Compteur=Valeur_initiale*)
2. Tester si la *Valeur_initiale* dépasse la *Valeur_finale* (du côté supérieur ou inférieur, selon la positivité ou la négativité du pas).
   Si oui, alors la boucle s'arrête et l'exécution se poursuit après le FINFAIRE
   Sinon,
   - Exécution du <Traitement>,
   - Incrémentation ou décrémentation de compteur par la valeur du pas (selon la positivité ou la négativité du pas),
   - Retour à l'étape 2.

En langage Pascal, la syntaxe de la boucle pour est donnée par :

For *compteur* **:=***valeur_initiale* **to** *valeur_finale* **do**
Begin
<traitement>
End ;

Remarques :

1- **Le traitement n'est mis entre « Begin » et « End » que s'il contient plus d'une instruction.**
2- **En Pascal, Si la *valeur_finale* > la *valeur_initiale* la syntaxe de la boucle « For » se présente comme suit :**

```
For compteur :=valeur_finale downto valeur_initiale do
Begin
<traitement>
End ;
```

3- **En langage Pascal, il n'est pas possible d'écrire une boucle « For » avec un pas différent de 1.**

Exemple 1 :

Ecrire un algorithme qui affiche dans l'ordre les valeurs de 1 à n (où n est un nombre entier positif).
Dans cet algorithme, il s'agit d'écrire les instructions permettant d'afficher les nombre de 1 à n. l'instruction d'affichage doit être répétée n fois.
Nous utilisons dans cet algorithme, la variable entière i pour compter le nombre de répétition. Cette variable est initialisée par la valeur 1 et incrémentée par 1 à chaque itération de la boucle. Donc, le pas de la boucle dans cet algorithme est égal à 1.

```
Algorithme exemple1 ;
Variable
n, i : entier ;
Début
Ecrire ('Donner un nombre entier positif') ;
Lire(n) ;
Pour  i de 1 à n pas 1 faire
   Ecrire (i) ;
Finfaire
Fin.
```

Exemple 2 :

Ecrire un algorithme qui calcule et affiche la moyenne de n nombres.
Dans cet algorithme, on doit lire n nombres, calculer leur somme et leur moyenne. L'opération de lecture doit se répétée n fois. Après chaque lecture, l'algorithme doit ajouter la valeur du nombre lu à la somme des nombres lus précédemment. Donc le traitement qui consiste à lire le nombre et à l'ajouter à la somme doit se répéter n fois.

```
Algorithme moyenne ;
Variable
i, n :entier ;
s, m, x : réel ;
Début
Ecrire ('Donner le nombre de nombres') ;
Lire (n) ;
s←0 ;
Pour  i de 1 à n pas 1 faire
  Ecrire ('Donner un nombre') ;
  Lire(x) ;
  s←s+x ;
finfaire
```

```
m←s/n ;
Ecrire ('La moyenne =',m) ;
fin.
```

2.2. La boucle Répéter

Cette structure permet de répéter le <Traitement> une ou plusieurs fois et de s'arrêter sur une condition. En effet, lorsque la condition est vérifiée, la boucle s'arrête, sinon elle réexécute le <Traitement>.

Syntaxe

```
Répéter
<Traitement>
Jusqu'à (condition d'arrêt)
```

Remarques

1. Dans cette boucle, le traitement est exécuté au moins une fois avant l'évaluation de la condition d'arrêt.
2. Il doit y avoir une action dans le <Traitement> qui modifie la valeur de la condition.

2.2.1. Les étapes d'exécution de la boucle Répéter

1. Exécution du <Traitement>
2. Tester la valeur de la <condition d'arrêt>
3. Si elle est vérifiée Alors la boucle s'arrête, Sinon Retour à l'étape 1

En langage Pascal, la syntaxe de la boucle Répéter est donnée par :

```
Repeat
<traitement>
Until (condition d'arrêt ) ;
```

Exemples

Reprendre les même exemples précédents avec la boucle Répéter.

A la différence de la boucle POUR, dans la boucle REPETER, il faut commencer par l'initialisation du compteur avant la boucle. De plus, à la fin du traitement, ce compteur doit être modifié (incrémenté ou décrémenté) afin d'atteindre la valeur finale.

Exemple 1 :

```
Algorithme exemple1 ;
Variable
n, i : entier ;
Début
Ecrire ('Donner un nombre entier positif non nul') ;
Lire(n) ;
i←1 ;
Répéter
   Ecrire (i) ;
   i←i+1 ;
jusqu'à (i>n) ;
Fin.
```

Exemple 2 :

```
Algorithme moyenne ;
Variable
i, n :entier ;
s, m, x : réel ;
Début
Ecrire ('Donner le nombre de nombres') ;
Lire (n) ;
s←0 ;
i←1 ;
Répéter
  Ecrire ('Donner un nombre') ;
  Lire(x) ;
  s←s+x ;
  i←i+1;
jusqu'à (i>n);
m←s/n ;
Ecrire ('La moyenne =',m) ;
fin.
```

2.3. La boucle TANT QUE

Cette structure permet de répéter le <Traitement> **zéro** ou plusieurs fois et de s'arrêter lorsque la condition d'exécution n'est plus vérifiée. En effet, lorsque la condition d'exécution est vérifiée, le <Traitement> est exécuté, sinon la boucle s'arrête.

Syntaxe

```
TANT QUE (condition d'exécution)
FAIRE
<Traitement>
FIN FAIRE
```

2.3.1. Les étapes d'exécution de la boucle Répéter

1. Tester de la valeur de la <condition d'exécution>
2. Si elle est vérifiée Alors
   Exécution du <Traitement>
   Retour à l'étape 1.

   Sinon Arrêt de la boucle.

Remarques

1. Dans cette boucle, le traitement peut ne jamais être exécuté, c'est lorsque la condition d'exécution est à fausse dés le départ.
2. Les paramètres de la condition doivent être initialisés par lecture ou par affectation avant la boucle.
3. Il doit y avoir une action dans le <Traitement> qui modifie la valeur de la condition.

En langage Pascal, la syntaxe de la boucle TANT QUE est donnée par :

```
While (condition d'exécution) do
begin
<Traitement>
End ;
```

Remarque :

Le traitement n'est mis entre « Begin » et « End » que s'il contient plus d'une instruction.

Exemples

Reprendre les même exemples précédents avec la boucle Tant que.

A la différence de la boucle REPETER, dans la boucle TANT QUE, la condition est évaluée au début.

A la différence de la boucle POUR, dans la boucle TANT QUE, il faut commencer par l'initialisation du compteur avant la boucle. De plus, à la fin du traitement, ce compteur doit être modifié afin d'atteindre la valeur finale.

Exemple 1 :

```
Algorithme exemple1 ;
Variable
n, i : entier ;
Début
Ecrire ('Donner un nombre entier positif') ;
Lire(n) ;
i←1 ;
Tant que (i<=n) faire
   Ecrire (i) ;
   i←i+1 ;
finfaire
Fin.
```

Exemple 2 :

```
Algorithme moyenne ;
Variable
i, n :entier ;
s, m, x : réel ;
Début
Ecrire ('Donner le nombre de nombres') ;
Lire (n) ;
s←0 ;
i←1 ;
Tant que (i<=n) faire
  Ecrire ('Donner un nombre') ;
  Lire(x) ;
  s←s+x ;
  i←i+1;
finfaire
m←s/n ;
Ecrire ('La moyenne =',m) ;
fin.
```

Remarques :

1. **Tout problème qui peut être résolu avec la boucle POUR, peut aussi être résolu avec les boucles REPETER et TANT QUE mais le contraire n'est pas vrai.**
2. **Si le nombre d'itération est connu à l'avance, il est préférable d'utiliser la boucle POUR.**

3. Si le traitement risque de ne pas s'exécuter, il est recommandé d'utiliser la boucle TANT QUE.

Exemple :

Ecrire un algorithme qui permet de :

- Lire une suite de lettres. La lecture se termine pas la saisi du '.'
- Afficher le nombre consonnes et le nombre voyelles.

Dans cet exemple, le nombre de lettres à lire n'est pas connu à l'avance. L'opération de lecture est répétée jusqu'à ce que le '.' soit introduit. Donc, il n'est pas possible d'utiliser la boucle POUR.

Avec la boucle TANT QUE

```
Algorithme  exemple3 ;
Variable
lettre : caractère ;
nbv, nbc : entier ;
Début
Nbv←0 ;
Nbc←0 ;
Ecrire('donner une lettre alphabétique') ;
Lire(lettre) ;
Tant que (lettre<>'.') faire
  Si (lettre='a' ou lettre= 'i' ou lettre='e' ou lettre='u' ou lettre='o' ou lettre='y') alors
     nbv←nbv+1
 sinon
  nbc←nbc+1 ;
finsi
Ecrire('donner une autre lettre alphabétique') ;
Lire(lettre) ;
Finfaire
Ecrire ('le nombre de voyellle=',nbv) ;
Ecrire ('le nombre de consonne=',nbc) ;
Fin.
```

La première opération dans cet algorithme consiste à initialiser les compteurs des voyelles et des consonnes. Afin de tester la condition de la boucle, il est nécessaire de faire une première lecture avant la boucle. Une autre lecture est nécessaire à l'intérieur de la boucle pour pouvoir réexécuter le traitement.

La boucle risque ne pas s'exécuter si on introduit le '.' comme première lettre. L'algorithme affiche dans ce cas la valeur zéro pour le nombre de voyelles et de consonnes.

Avec la boucle REPETER

```
Algorithme  exemple3 ;
Variable
lettre : caractère ;
nbv, nbc : entier ;
Début
Nbv←0 ;
Nbc←0 ;
Ecrire('donner une lettre alphabétique') ;
Lire(lettre) ;
```

```
Si (lettre<>'.') alors
Répéter
  Si (lettre='a' ou lettre= 'i' ou lettre='e' ou lettre='u' ou lettre='o' ou lettre='y') alors
     nbv←nbv+1
 sinon
  nbc←nbc+1 ;
finsi
Ecrire('donner une autre lettre alphabétique') ;
Lire(lettre) ;
Jusqu'à (lettre='.') ;
finsi
Ecrire ('le nombre de voyellle=',nbv) ;
Ecrire ('le nombre de consonne=',nbc) ;
Fin.
```

Le même principe s'applique pour la boucle REPETER. Le test avant la boucle permet de gérer le cas ou on commence par introduire le '.'

Si ce test n'est pas utilisé, la boucle s'exécute même si on commence par introduire le '.' comme première lettre et l'algorithme affiche dans ce cas le nombre de consonne = 1 (au lieu d'afficher 0).