## Les instructions itératives (les boucles)

###### Introduction

Considérons le même exemple qu’on a vu précédemment concernant le calcul de la moyenne générale d’un étudiant. Pour se faire, on doit :

* + Lire toutes les notes (et leurs coefficients) de l’étudiant,
  + Calculer la somme des notes,
  + Diviser la somme obtenue sur le nombre (ou sur la somme des coefficients).

Si l’on veut maintenant calculer la moyenne d’un autre étudiant, les mêmes instructions doivent être répétées.

Pour N d’étudiants, il nous faudra donc répéter N fois la même séquence d’instructions. Cet exemple soulève deux questions importantes :

1. Comment éviter d’écrire plusieurs fois la même séquence d’instructions ?
2. Combien de fois doit-on répéter l’exécution de la séquence d’instructions pour obtenir le résultat attendu ?

Pour répondre à ces questions, de nouvelles instructions de contrôle sont introduites. Il s’agit des instructions itératives (appelées aussi les boucles ou les itérations).

###### Définition

Une boucle (ou itération) est une instruction de contrôle qui permet de répéter plusieurs fois un ensemble d’instructions. Généralement, deux cas sont distingués :

* Le nombre de répétitions est connu.
* Le nombre des répétitions est inconnu ou variable.

###### L’instruction « Pour »

Lorsque le nombre de répétitions est déterminé (connu), l’utilisation de l’instruction « Pour » est privilégiée. Une structure de boucle avec l’instruction « Pour » s’arrête une fois que le nombre de répétitions est atteint. Cette structure possède un indice (compteur) de contrôle d’itérations caractérisé par :

* une valeur initiale,
* une valeur finale,
* un pas de variation.

###### Syntaxe :

**Pour** indice **de** début **à** fin **Pas** valeur\_du\_pas instruction(s) ;

###### Fin Pour

Cette structure est dite « croissante » lorsque la valeur initiale de l’indice est inférieure à sa valeur finale, le pas de variation est par conséquent positif**.** Autrement, elle est dite

« décroissante ».

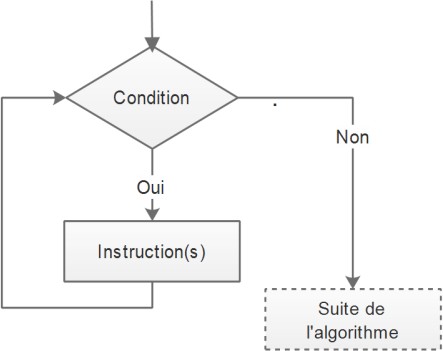
###### Exemple : un compteur croissant/décroissant

Les deux algorithmes suivants comptent de 1 à N et de N à 1 respectivement.

|  |  |
| --- | --- |
| **Algorithme** compteur\_croissant ;  **Var** i : entier ; **Const** N=100 ; **Début**  **Pour** i **de** 1 **à** N **/\*** *par défaut le pas = 1* \*/  **Ecrire** (i);  **FinPour Fin**  Résultat d’exécution **:** 1,2, 3, … , 99, 100 | **Algorithme** compteur\_decroissant ;  **Var** i : entier ; **Const** N=100 ; **Début**  **Pour** i **de** N **à** 1 **/\*** *par défaut le pas = -1* \*/  **Ecrire** (i);  **FinPour Fin**  Résultat d’exécution : 100, 99, 98, … , 2, 1 |

**Remarque** : Si la valeur du « pas » n’est pas précisée dans l’instruction « Pour », elle est par défaut égale à un (1).

###### L’instruction « Tant que… faire »

Cette instruction permet de tester une condition et répéter le traitement associé tant que cette condition est vérifiée.

###### Syntaxe:

**Tant que** condition **faire**

instruction(s) ;

###### FinTq

**Exemple** : Réécrivons l’algorithme précédent avec cette instruction.

**Var** i : entier ;

**Début**

i ← 1 ;

**Tant que (**i<=100) **faire**

**Ecrire** (i) ; i ← i+1 ;

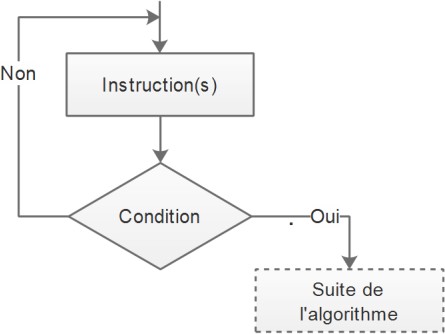
**FinTq**

**Fin**

###### L’instruction « Répéter… jusqu’à »

Dans cette instruction, un traitement est exécuté au moins une fois puis sa répétition se poursuit jusqu’à ce que la condition soit vérifiée.

###### Syntaxe:

**Répéter**

instruction(s) ;

**Jusqu’à** (condition) ;

**Exemple** : Soit l’algorithme suivant :

**Var** n, p : entier ;

**Début**

**Répéter**

**Ecrire** ("Donner un nombre :") ; **Lire (**n) ;

**p** ← n\*n ; **Ecrire** (p);

**Jusqu’à (**n=0)

**Ecrire** ("Fin de l’algorithme") ;

**Fin**

Les instructions encadrées par les mots **répéter** et **jusqu’à** constituent le bloc de la boucle qu’il faut répéter jusqu’à ce que la condition (**n=0)** soit vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l’utilisateur.

**Question ?**

*Réécrire l’algorithme précédent avec « Tant que… faire » puis avec « Pour ».*

###### Remarque :

Dans la boucle « Répéter… jusqu’à », la condition telle qu’elle est exprimée ci-dessus, constitue une **condition d’arrêt** de la boucle ; mais réellement, cela diffère selon le langage de programmation utilisé. Par exemple, en Pascal, la condition de cette boucle est une condition d’arrêt. Alors qu’en langage C, cette condition est exprimée en tant qu’une **condition de continuation**.

###### La notion du compteur

Un compteur est une variable associée à la boucle dont la valeur est incrémentée de un à chaque itération. Elle sert donc à compter le nombre d’itérations (répétitions) de la boucle. La notion du compteur est associée particulièrement aux deux boucles : « **Répéter…jusqu’à** » et « **Tant que…faire** ». Par contre, dans la boucle

« **Pour** », c’est l’indice qui joue le rôle du compteur.

L’utilisation du compteur dans les deux premières boucles est exprimée ainsi :

*Bloc de la boucle*

compt ← 0 ;

###### Répéter

instruction(s) ;

…

compt ← compt +1 ;

**Jusqu’à (**condition) ;

compt ← 0 ;

**Tant que** (condition) **faire**

instruction(s) ;

…

compt ← compt +1 ;

###### FinTant que ;

**Remarque** :

Il faut toujours initialiser le compteur avant de commencer le comptage. La variable

« compt » (utilisée ci-dessus comme compteur), a été initialisée à zéro (0) avant le début de chaque boucle.

L’instruction « compt ← compt +1 » incrémente la valeur de « compt » de un (1). Elle peut être placée n’importe où à l’intérieur du bloc de la boucle.

###### Exemple :

|  |  |
| --- | --- |
| i ← 0 ;  **Répéter**  **Ecrire** (i); i ← i +1 ;  **Jusqu’à (**i=5) ; | i ← 0 ;  **Tant que** (i<5) **faire Ecrire** (i);  i ← i +1 ;  **FinTant que ;** |
| Résultat d’exécution **:** 0, 1, 2, 3, 4 | Résultat d’exécution **:** 0, 1, 2, 3, 4 |

###### La notion d’accumulation

Cette notion est fondamentale en programmation. Elle est utilisée notamment pour calculer la somme d’un ensemble de valeurs. L’instruction correspondante se présente ainsi :

variable ← variable + valeur ;

Cette instruction consiste à ajouter une valeur à une variable numérique, puis affecter le résultat dans la variable elle-même. En d’autres termes, la nouvelle valeur de variable égale à l’ancienne plus une certaine valeur [4].

**Exemple** : calcul de la somme de n valeurs données par l’utilisateur :

**Var** i, n : **entier** ; som, val : **réel** ;

**Début**

**Écrire** ("Donner le nombre de valeurs :") ; **Lire** (n) ; som ← 0 ;

**Pour** i **de** 1 **à** n

**Écrire** ("Enter une valeur :") ; **Lire** (val) ; som ← som + val ;

**Finpour**

**Écrire** ("La somme des valeurs est égale à :", som) ;

**Fin**

###### Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Deux ou plusieurs boucles imbriquées peuvent être aussi les mêmes ou différentes.

###### Exemple :

**Pour** i **de** 1 **à** 2

**Écrire** ("i = ", i) ;

**Pour** j **de** 1 **à** 3 *boucle 1*

**Écrire** ("j = ", j) ; *boucle 2*

**Finpour Finpour**

Dans l’exemple ci-dessus, chaque itération de la boucle extérieure (boucle 1) exécute la boucle intérieure (boucle 2) jusqu’à la fin avant de passer à l’itération suivante, et ainsi de

suite jusqu’à la fin des deux boucles. Ainsi, le résultat d’exécution peut être représenté comme suit :

###### Remarque :

**i = 1**

j = 1

j = 2

j = 3

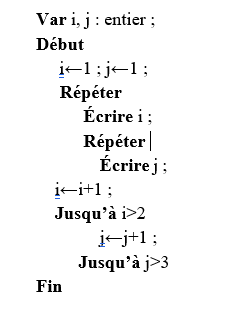
###### i = 2

j = 1

j = 2

j = 3

Des boucles peuvent être imbriquées ou successives. Cependant, elles ne peuvent jamais être croisées. Par exemple, l’algorithme suivant est faux puisqu’il comporte deux boucles croisées :



1. **Conclusion**

Ce chapitre a été consacré aux structures itératives ou boucles qui permettent de répéter l’exécution d’une séquence d’instructions plusieurs fois selon un nombre fixe ou certains critères dont l’utilisation a été explicitée à travers différents exemples. Ainsi, ces instructions sont d’une grande importance dans la manipulation de certaines structures de données telles que les tableaux que nous aborderons dans le prochain chapitre.