Chapitre 4

1. Introduction

Une boucle est une instruction itérative (répétitive) composée d'une ou plusieurs autres instructions qui peuvent être exécutées plus d'une fois, en fonction du résultat d'une condition. On appelle chacune de ces exécutions potentielles une itération (d'où le nom instructions itératives).
On a trois types de boucles: "**TANTQUE**", "**REPETER**", Boucle "**POUR**".

2. La boucle Tant que

Définition

Est une instruction itérative composée d'une condition et d'un bloc d'instructions (corps de la boucle), le bloc d'instruction est exécuté zéro ou plusieurs fois selon la condition.

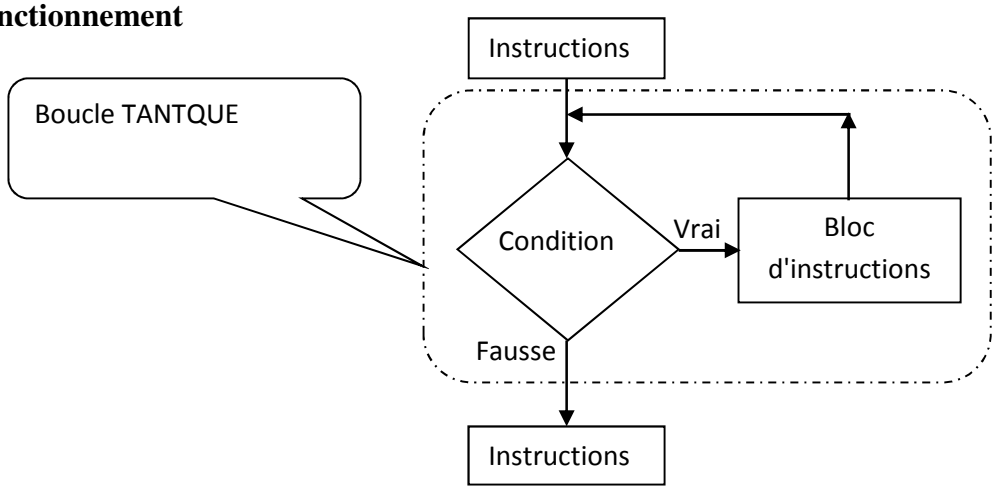
Remarques

- La boucle " **TANTQUE** " est la boucle la plus simple et la plus générale.
- la condition est évaluée avant chaque itération.

Syntaxe

TANTQUE (<Condition>)
<Bloc d'instructions>
FINTANTQUE

Fonctionnement



Après l'évaluation de la condition, si la condition est vrai, le bloc d'instructions est exécuté et on retourne au test pour une nouvelle évaluation (ont dit qu'on boucle) jusqu'à ce que l'évaluation de la condition donne une valeur "FAUX", une fois que la condition est fausse on poursuit l'exécution des instructions après le mot "FIN TAN TQUE"

Exemple

Age ← 0
TANTQUE age < 18
Age ← age +1
FIN TANTQUE
ECRIRE "Adulte"
Traduction en C

```
Age=0 ;
While (age<18)
{ Age++; }
```

Printf("%d",age);

Remarques

1. Le nombre d'itérations dans une boucle " **TANTQUE** " n'est pas connu au moment d'entrée dans la boucle. Il dépend de la valeur d'évolution de la condition.
2. Le bloc d'instructions d'une boucle " **TANTQUE** " peut ne jamais être exécuté (si la condition et fausse pour la première vérification).

   Exemple :

   i ← 0
   TANTQUE i > 0
   i← i+1
   FINTANTQUE
3. On peut ne jamais sortir d'une boucle " **TANTQUE** " (condition toujours vérifiée). Pour éviter ça une des instructions du corps de la boucle doit absolument changer la valeur de la condition de vrai à faux (après un certain nombre d'itérations).

   Exemple:

   i ← 0
   TANTQUE i > 0
   i← i+1
   FINTANTQUE

3. La Boucle "REPETER"

Définition

Est une instruction itérative composée d'un bloc d'instructions (corps de la boucle) et d'une condition, le bloc d'instruction est exécuté une ou plusieurs fois et leur exécution est répétée jusqu'à ce que la condition soit vraie.

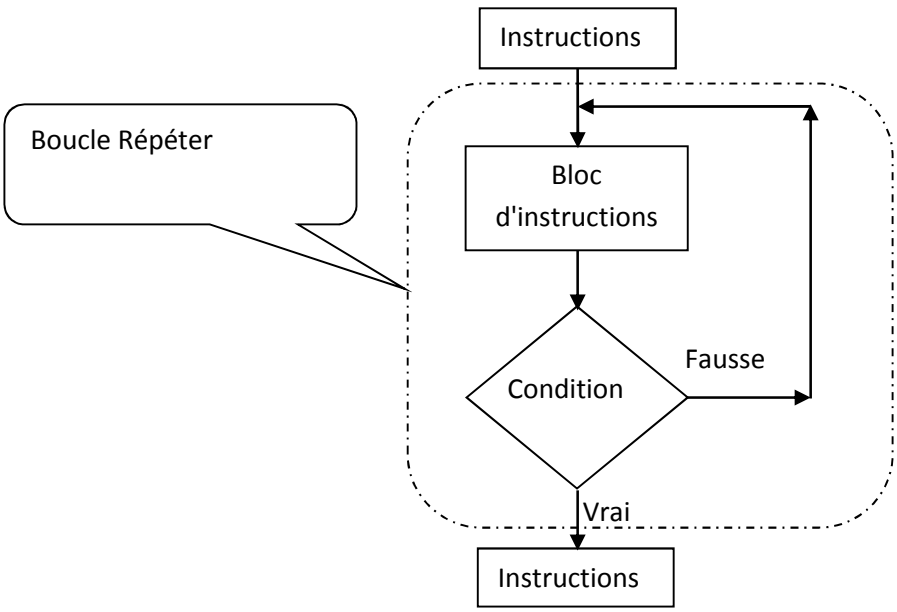
Remarques

La condition est évaluée après l'exécution du bloc d'instruction de la boucle " **REPETER** "

Syntaxe

REPETER
<Bloc d'instructions>
JUSQU'A <Condition>

Fonctionnement



Après l'exécution du bloc d'instructions, la condition est évaluée, si la condition est fausse, on retourne à une nouvelle exécution du bloc d'instructions jusqu'à ce que l'évaluation de la condition donne une valeur "vrai", une fois que la condition est vrai on poursuit l'exécution des instructions après le mot "**JUSQU'A**"

Exemple

```
Age ← 0
REPETER
        Age ←  age +1
JUSQU'A age >= 18
ECRIRE "Adulte"
```

Remarques

1. Le nombre d'itérations dans une boucle " **REPETER** " n'est pas connu au moment d'entrée dans la boucle. Il dépend de la valeur d'évolution de la condition.
2. On peut ne jamais sortir d'une boucle " **REPETER** " (condition toujours fausse). Pour éviter ça une des instructions du corps de la boucle doit absolument changer la valeur de la condition de fausse à vrai (après un certain nombre d'itérations).

   Exemple

   ```
   i ← 0
   REPETER
           i← i+1
   JUSQU'A  i < 0
   ```

Cette structure n'existe pas en c mais il existe l'instruction do …while qui permet de réaliser une action ou une suite d'actions tant que la condition de branchement est réalisée. Elle est réalisée au moins une fois !

Structure en C :

```
I=0 ;
Do {
I+=1 ;
}While(I<18)
```

4. Boucle "POUR"

Définition

Est une instruction itérative composée d'un bloc d'instructions (corps de la boucle) et d'un compteur pour compter les itérations, le bloc d'instruction est exécuté un nombre de fois connu à l'avance. Il est nécessaire d'indiquer la première et la dernière valeur du compteur (valeur initiale et valeur finale du compteur).

Compteur

Est une variable de type entier ou caractère. Elle doit être déclarée et initialisée.

Syntaxe

```
POUR compteur ←  valeur_initiale A valeur_finale PAS valeur_du_pas
 <Bloc d'instructions>
FIN POUR
```

Pas

Est un entier qui peut être positif ou négatif. Le "Pas" peut ne pas être mentionné, car par défaut sa valeur est égale à 1. Dans ce cas, le nombre d'itérations est égal à valeur_finale - valeur_initiale + 1.

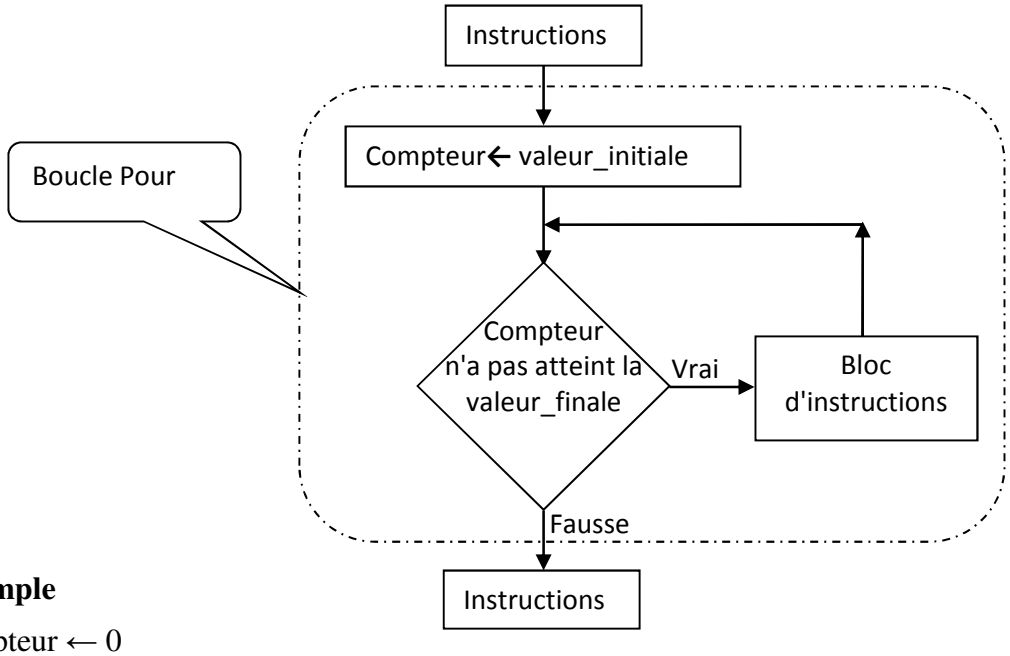
Valeur_initiale et valeur_finale

Peuvent être des valeurs, des variables initialisées avant le début de la boucle ou des expressions de même type que le compteur**.**

Fonctionnement

D'abord, on doit initialiser le compteur par une valeur initiale, Après le test est-ce-que le compteur arrive à sa valeur finale ou non, si le compteur n'arrive pas à sa valeur finale, le bloc d'instructions est exécuté et on refait le test du compteur jusqu'à ce que le compteur arrive à sa valeur finale, une fois le compteur arrive à sa valeur finale on poursuit l'exécution des instructions après le mot "**FIN POUR**".

Le compteur est incrémenté par la valeur du pas automatiquement.



Exemple

compteur ← 0
POUR compteur ← 0 **A** 10
    x ← x+2
    ECRIRE x, compteur
FIN POUR

Remarques

1. Le nombre d'itérations dans une boucle "**POUR**" est connu avant le début de la boucle
2. Il faut éviter de modifier la valeur du compteur (et de valeur_finale) à l'intérieur de la boucle. Une telle modification:
    - Perturbe le nombre d'itérations de la boucle "**POUR**".
    - Présente le risque d'aboutir à une boucle infinie.

    Exemple

    POUR i ← 1 **A** 5
        i ← i -1
    ECRIRE " la valeur de i est = ", i
    FINPOUR
3. La boucle "**POUR**" est un cas particulier de "**TANTQUE**". Tout ce qu'on peut écrire avec "**POUR**" peut être remplacé avec " **TANTQUE** " mais l'inverse est faux.
    Le code de la boucle "**POUR**" peut être remplacé par le code suivant:
        compteur ← valeur_initiale
        TANTQUE compteur <= valeur_finale

<Bloc d'instructions>
compteur ← compteur + **PAS**
FINTANTQUE

Structure en C :

```
For (Y=5000 ; Y>=0 ; Y=Y-1)
{
B=B+1 ;
}
```

5. Boucles imbriquées

De même que l'instruction conditionnelle "**SI SINON**" les boucles peuvent contenir d'autres boucles.

Exemple 1: boucles non imbriquées (séquentielles)

```
VAR i, j ENTIER
POUR i ← 1 A 10
        ECRIRE "Première boucle"
        ECRIRE "Première boucle"
FINPOUR

POUR j ← 1 A 5
        ECRIRE "Deuxième boucle"
FINPOUR
```

La première boucle exécute toutes ses itération et affiche dix fois le message "Première boucle" et ensuite la deuxième boucle s'exécute et affiche cinq fois le message "Deuxième boucle".

Exemple 2: boucles imbriquées

```
VAR i, j ENTIER
POUR i ← 1 A 10
        ECRIRE ("Première boucle")
        POUR j ← 1 A 5
                ECRIRE "Deuxième boucle"
        FINPOUR
ECRIRE "Première boucle"
FINPOUR
```

Pour chaque itération de la première boucle, le programme écrira une fois "Première boucle" puis cinq fois "Deuxième boucle", puis une fois "Première boucle". A la fin, il y aura donc eu 10 x (5+2) = 70 messages affichés.

6. Choix d'un type de boucle

- **Choix entre la boucle "POUR" et les autres boucles:**

  Si le nombre d'itérations est connu avant l'exécution de la boucle, il est préférable d'utiliser la boucle "**POUR**", sinon, on fera appel à l'une des boucles "**TANTQUE**" ou "**REPETER**".

- **Choix entre les boucles " TANTQUE " et " REPETER ":**

  Si on doit tester la condition avant de commencer l'exécution des instructions de la boucle, on utilisera " **TANTQUE** ", mais si la valeur de la condition dépend d'une première exécution des instructions de la boucle, on utilisera " **REPETER** ".

7. Exercice

On veut écrire un algorithme qui calcul la somme des entiers positifs inférieurs ou égaux à un entier donnée N.

Résolution

1ère étapes: Analyse du problème

Les données : la valeur de N

Les cas possible :

- 1er cas : N = 0 ➔ afficher 0
- 2eme cas : N < 0 ➔ afficher 'erreur'
- 3eme cas : N > 0 ➔ On calcule la somme des N premiers entiers positifs

Les résultats : Affichage de la somme calculée ou le message d'erreur.

2ème étapes : Conceptions et démarches de résolution

```
{Déclaration des variables utilisées}
VAR N, i, somme : ENTIER
{Lecture des données}
ECRIRE donner la valeur de N
LIRE N
Test sur les cas possibles:
SI  N<0 ALORS
        ECRIRE "Erreur"
SINON
        SI  N= 0 ALORS
                ECRIRE "La somme = 0"
        SINON
                {Initialisation du compteur de la boucle}
                i ← 1
                {Initialisation de la variable somme par 0}
                somme ← 0
                {La boucle qui calcule la somme}
                PUOR  i ←  1 A N
                        somme ← somme + i
                FIN POUR
                {Affichage de la somme calculée}
                ECRIRE "la somme des entiers positifs inférieur à", N, "est", somme
        FINSI
FINSI
FIN
```

Table des matières