ALGORITHMIQUE ET PROGRAMMATION INFORMATIQUE

Chapitre 4 : Les Structures répétitives ou itératives

Les structures répétitives ou boucles permettent l'exécution d'une suite d'instructions soit

- un nombre de fois défini à l'avance ;
- jusqu'à ce qu'une condition ait une valeur donnée ou qu'elle soit vérifiée.

Une exécution des instructions à répéter s'appelle tour de boucle ou itération.

I. Les principales boucles

1. Boucle finie

a. La boucle POUR

i. Définition

La boucle POUR permet de répéter une suite d'instructions un nombre de fois connu (chaque élément d'une liste) à l'avance.

ii. syntaxe

POUR <cpt> **DE** <debut> **A** <fin> [PAR PAS DE <pas>] **FAIRE**

<bloc>

FINPOUR

Ou

POUR <cpt> **DE** <debut> **A** <fin> [PAR PAS DE <pas>]

<bloc>

Cpt SUIVANT

<cpt> est une variable entière, appelée variable d'itération ou compteur, permettant de distinguer les itérations les unes des autres.

<debut> est la première valeur de <cpt> et <fin> sa dernière valeur.

<pas> est le pas de la boucle, il permet de déterminer l'itération suivante. Par défaut le pas vaut 1.

iii. Fonctionnement

Etape 1 : La variable d'itération<cpt> est initialisée à la valeur <debut>;

Etape 2 : La condition <cpt> ≤ <fin> est évaluée ;

Etape 3 : Si la condition est vraie, la suite d'instructions <bloc> est exécutée, la valeur de <cpt> est augmentée de <pas> et retour à l'étape 2;
Etape 4 : Si la condition est fausse, on sort de la boucle.

La boucle Pour peut se présenter sous différentes formes:

1. Pour i dans {1..100} Faire …
2. Pour i allant de 1 à 100 Faire …
3. Pour i←1 à 100 Faire …
4. Pour i=1 à 100 Faire …
5. Pour i=1 à 100 de 3 en 3 Faire …
6. Pour i=1 à 100 de -1 en -1 Faire …
7. …

iv. Exemple :

Ecrire un algorithme qui affiche les n premiers entiers.

v. Remarque :

- Le nom de la variable d'itération doit être un identificateur valide c'est-à-dire une variable déclarée.
- On ne doit pas modifier la variable d'itération à l'intérieur de la boucle.
- Les instructions de la boucle ne sont jamais exécutées si :
- Le pas est positif et la valeur initiale est supérieure à la valeur finale de la boucle ;
- Le pas est négatif et la valeur initiale est inférieure à la valeur finale de la boucle.

vi. Sens de la boucle

Une boucle POUR est croissante si son pas est positif. Dans une boucle croissante les valeurs que prend successivement le compteur sont énumérées dans l'ordre croissant.
Une boucle POUR est décroissante si son pas est négatif. Dans une boucle décroissante les valeurs que prend successivement le compteur sont énumérées dans l'ordre décroissant.
La notation des boucles décroissantes de pas -1 peut être comme suit :

2. Boucle infinie

a. La boucle TANT QUE

i. Définition

La boucle TANT QUE permet de répéter une suite d'instructions tant qu'une condition est vraie.

ii. syntaxe :

TANT QUE (<condition>) **FAIRE**

<bloc>

FINTQ

iii. Fonctionnement

Etape 1 : la condition est évaluée ;

Etape 2 : si la condition est fausse : on quitte la boucle tant que ;

Etape 3 : si la condition est vraie, on exécute<bloc> puis on remonte à l'étape 1 tester de nouveau la condition.

Le déroulement pas à pas de cette instruction équivaut au schéma suivant :

SI(<condition>) ALORS

<bloc>

SI(<condition>) ALORS

<bloc>

SI(<condition>) ALORS

<bloc>

Le nombre de répétitions n'est pas connu explicitement, il dépendra des données fournies.

iv. Remarque :

- La condition de la boucle TANT QUE est testée avant l'exécution des instructions ; les instructions de la boucle TANT QUE peuvent donc ne jamais être exécutées. La boucle TANT QUE permet en réalité de répéter un traitement, 0 ou plusieurs fois.
- On doit s'assurer que la condition a une valeur déterminée avant le début de la boucle.
- La condition doit à un moment donné devenir fausse pour la boucle s'arrête ; aussi la suite d'instructions à répéter doit donc modifier les variables intervenant dans la condition afin de la conduire à prendre la valeur .faux.

b. La boucle REPETER JUSQU'A

i. Définition

La boucle REPETER JUSQU'A sert à répéter une suite d'instructions jusqu'à ce qu'une condition soit vraie.

ii. syntaxe

REPETER

<bloc>

JUSQU'A (<condition>)

iii. Fonctionnement

Etape 1 : on exécute la suite d'instructions ;

Etape 2 : on évalue la condition ;

Etape 3 : si la condition est vraie, on quitte la boucle répéter ;

Etape 4 : si la condition est fausse, on recommence.

iv. Remarque :

- La condition de la boucle REPETER JUSQU'A est une condition d'arrêt : on sort de la boucle lorsque la condition est vraie.
- Les instructions à répéter sont exécuter avant l'évaluation de la condition ; donc les instructions de la boucle sont exécutées au moins une fois.
- Les instructions de la boucle REPETER JUSQU'A doivent modifier les variables intervenant dans la condition afin de la conduire à prendre la valeur .vrai.

c. La boucle REPETER TANT QUE

i. Définition

La boucle REPETER TANT QUE est une variante de la boucle REPETER JUSQU'A dans laquelle des instructions sont exécutées jusqu'à ce qu'une condition soit fausse.

ii. Syntaxe

REPETER

<bloc>

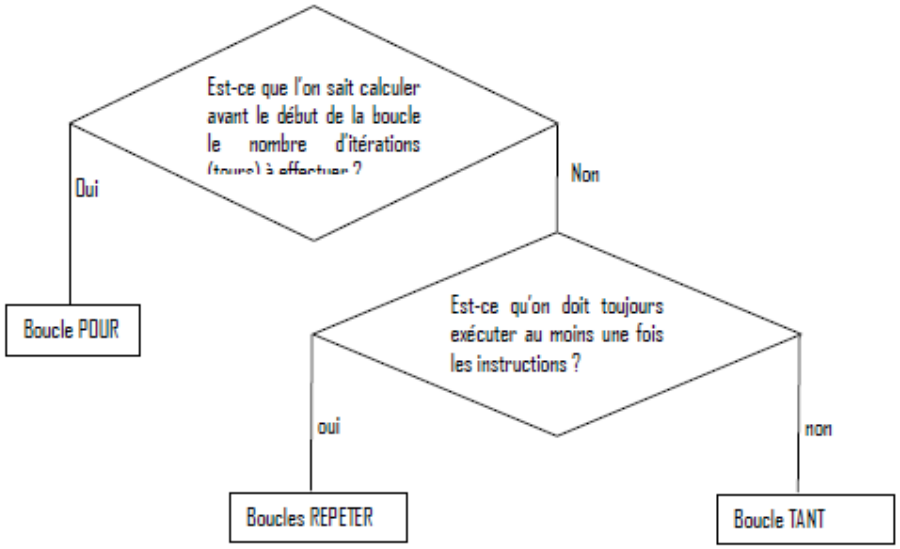
TANT QUE(<condition>)

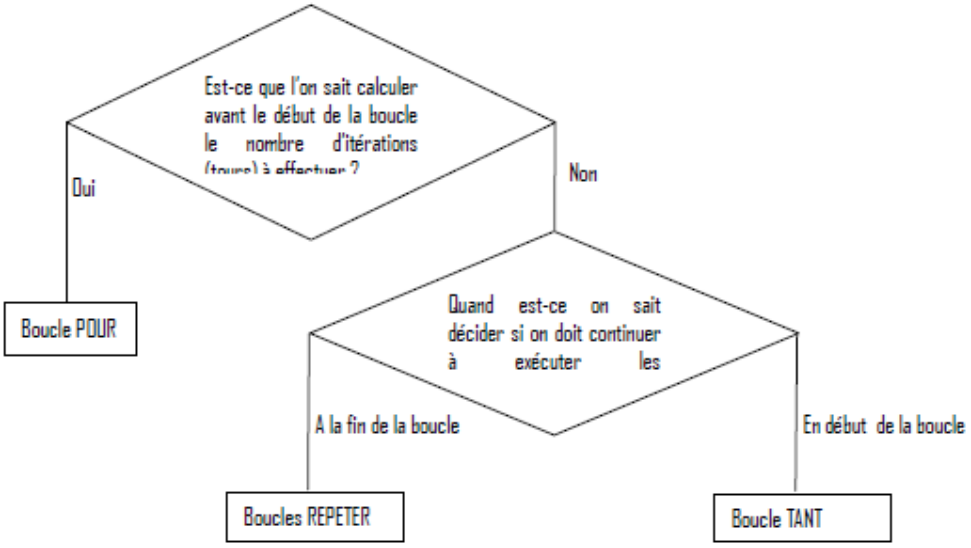
iii. Remarque

La condition de la boucle REPETER TANT QUE est une condition de continuation : elle doit être vraie pour que l'on poursuive l'exécution de la boucle.

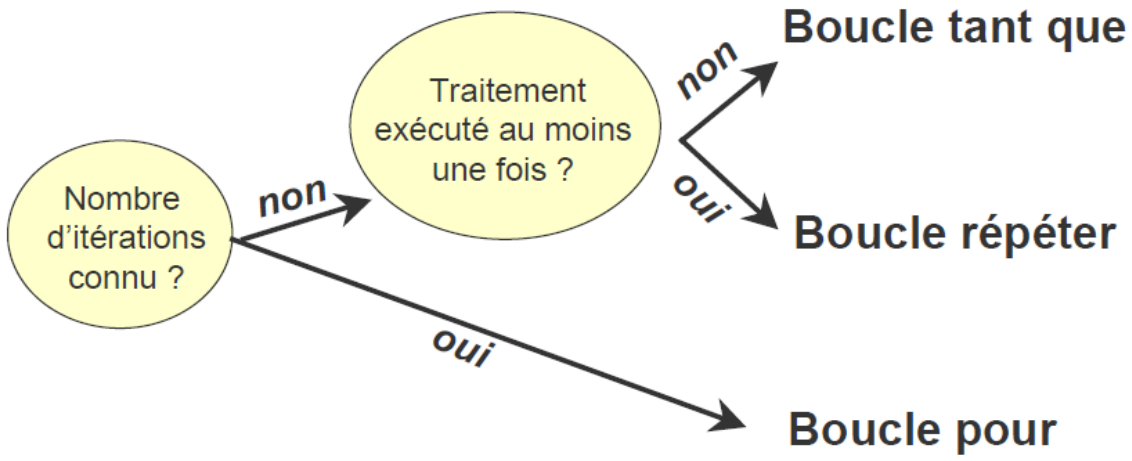
II. Comment choisir un type de boucle

Lorsqu'on a une suite à répéter, on peut utiliser les deux questionnaires équivalents suivants pour choisir la boucle qui convient le mieux.





Remarque : Si le nombre d'itération connu à l'avance on utilise POUR mais si la boucle doit s'arrête sur événement particulier on opte pour TANT QUE ou REPETE JUSQU'A.



III. Comparaison de boucles

a. Comparaison boucles pour et tant que

pour cpt de 1 **à** nbVal **faire**

Ecrire("Donnez une valeur :")

Lire(valeur)

totalValeurs←totalValeurs+ valeur *{cumul}*

finpour

C'est équivaut à:

cpt ←0

tant que cpt <nbVal **faire**

Ecrire("Donnez une valeur :")

Lire(valeur)

totalValeurs←totalValeurs+ valeur *{cumul}*

cpt ←cpt + 1 *{compte le nombre de valeurs traitées}*

fintq

Dans cette comparaison il ressort que :

- **Implicitement, l'instruction pour:**
    - initialise un compteur
    - incrémente le compteur à chaque pas
    - vérifie que le compteur ne dépasse pas la borne supérieure
- **Explicitement, l'instruction tant que doit**
    - initialiser un compteur
    - incrémenter le compteur à chaque pas
    - vérifier que le compteur ne dépasse pas la borne supérieure

b. Comparaison boucles répéter et tant que

Répéter

Ecrire("Donnez une valeur positive paire :")

Lire(valeur)

tant que (valeur < 0 **ou**(valeur % 2)

C'est équivaut à:

afficher("Donnez une valeur positive paire :")

saisir(valeur)

tant que(valeur < 0 **ou**(valeur % 2) ≠0) **faire**

Eccrire("Donnez une valeur positive paire:")

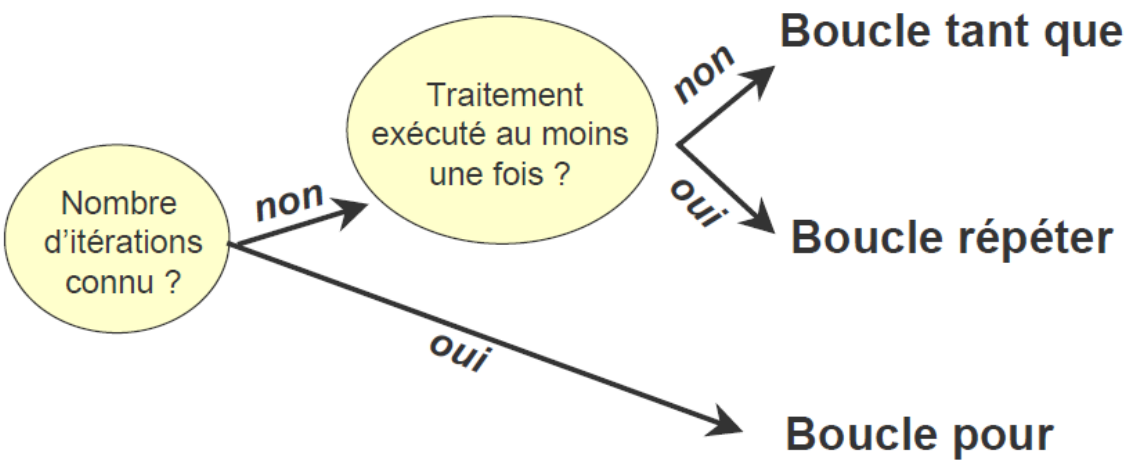
Lire(valeur)

Fintq

Dans cette comparaison il ressort que :

- **Boucle tant que**
    - condition vérifiée **avant** chaque exécution du traitement
    - le traitement peut donc ne pas être exécuté
    - de plus : la condition porte surtout sur la saisie de nouvelles variables (relance)
- **Boucle répéter tant que**
    - condition vérifiée **après** chaque exécution du traitement

- le traitement est exécuté au moins une fois
- de plus : la condition porte surtout sur le résultat du traitement



c. **Remarque**

- la boucle répéter est typique pour les saisies avec vérification.
- finsi, fintq et finpour peuvent être omis si le corps se limite à une seule instruction

  Exemples:

  Si val >0 **alors** ecrire(«fini!»)

  Pour i de 1 à MAX **faire** ecrire(i ×val)

Exercice 1 : Afficher le carré des valeurs saisies tant qu'on ne saisit pas 0

Réponse :

```
lire(val)
tant que val ≠0 faire
        ecrire(val ×val)
        lire(val)
fintq
```

avec la boucle répéter on a :

```
repeter
        lire(val)
        ecrire(val ×val)
tant que val ≠0
```

Exercice 2 : Saisir des données et s'arrêter dès que leur somme dépasse 500

Réponse :

```
lire(val)
somme ←val
tant que somme ≤500 faire
        lire(val)
        somme ←somme + val
fintq
```

Avec la boucle répéter on a :

```
somme ←0
répéter
        lire(val)
        somme ←somme + val
tant que somme ≤500
```

TD 3

Exercice 1

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Exercice 2

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

Exercice 3

Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

1 + 2 + 3 + 4 + 5 = 15

NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

Exercice 4

Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée 8 !, vaut

1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

Exercice 5

Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14
etc.

Entrez le nombre numéro 20 : 6
Le plus grand de ces nombres est : 14
Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :
C'était le nombre numéro 2

Exercice 6

Réécrire l'algorithme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice 7

Lire la suite des prix (en euros entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et simuler la remise de la monnaie en affichant les textes "10 Euros", "5 Euros" et "1 Euro" autant de fois qu'il y a de coupures de chaque sorte à rendre.

Exercice 8

Ecrire un algorithme qui demande à l'user un nombre N, puis calcule la somme des nombres de 0 à N

Exemples

N = 6

Somme = 0+1+2+3+4+5+6 = 21

Exercice 9

Ecrire un algorithme qui demande un nombre puis vérifier si ce nombre est premier ou non.

Exercice 10

Ecrire un algorithme pour calculer la somme des n premiers termes de la suite suivante :

Un =4+2n/3n

U0=1

N = 4

Somme = U0+U1+U2+U3+U4

1+ (4+2)/3+ (4+4)/6+……

Exercice 11

Ecrire un algorithme qui demande à l'user un nombre et :

Affiche les diviseurs de ce nombre

Le nombre de ces diviseurs

La somme des diviseurs de ce nombre