Chapitre 4 : Les structures de répétition (boucles)

1. Introduction :

- La structure de répétition est une des trois structures fondamentales qui sont :
  1) Actions séquentielles,
  2) Actions conditionnelles,
  3) Actions de répétition.

2. Actions de répétition (itérative) :(boucles)

- Les itérations (boucles) permettent de répéter plusieurs fois une même série d'instructions.
- Il y a principalement deux types de boucles :
  - La boucle qui permet de répéter des instructions jusqu'à une condition d'arrêt, il s'agit de la boucle **Tant que** et **répéter**.
  - La boucle qui permet de répéter des instructions un certain nombre de fois, il s'agit de la boucle **Pour**

2.1. La boucle « Tant que » ou « while » en langage C++ :

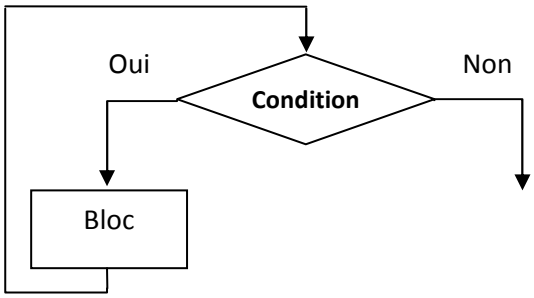
- Elle permet la répétition d'une ou plusieurs actions ***tant que la condition reste vérifiée.***

Syntaxe :

Tant que (condition) **Faire**

< Bloc d'actions >

Fin tant que



- Il s'agit de répéter l'exécution des instructions du **< Bloc d'actions >** tant que la condition **est vérifiée** et arrêter leur exécution dès que la condition **devient non vérifiée**.

Exemple :

Ecrire un algorithme qui calcule la somme des **n** premiers nombres entiers positifs ?

Algorithme som-ent1
    n, i, som : entier;
Début
Lire(n) ; /* pour connaitre où on va s'arrêter*/
Som← 0 ; /* la somme est initialisée à 0*/
i←0 ; /* il faut préciser la valeur initiale de i */
Tantque (i ≤ n) **Faire**
    Som←Som + i ; /* la nouvelle valeur de Som reçoit l'ancienne valeur + la valeur actuelle de i (dans cette itération)*/

i←i + 1; /* incrimination de i par 1 à chaque itération*/

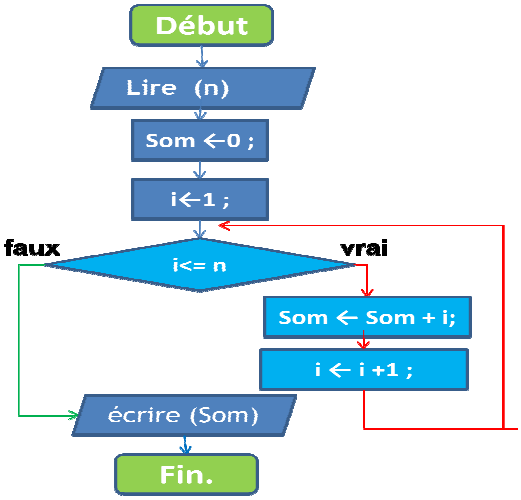
Fin tan que

Ecrire (som) ;

Fin.

Remarque :

- L'action répétitive **Tantque** est utilisée quand on ne connaît pas à priori le nombre de répétitions et lorsqu'il est possible de ne pas du tout exécuter les actions du bloc ; dans l'exemple précédant, si l'utilisateur saisie une valeur négative à n (-3 par exemple) le bloc à l'intérieur de la boucle tant que ne s'exécute jamais.
- L'organigramme de l'exemple précédant est comme suit :



Propriétés

- On sait qu'à la sortie de la boucle, la condition de la boucle est fausse.
- On ne connait pas le nombre d'itérations à effectuer, mais à chaque itération, on vérifie si la condition est vraie ou fausse.

2.1.1 Exercice : (Distributeur automatique du café)

On vous demande d'écrire un algorithme qui pose à l'utilisateur la question suivante: « voulez vous un café ? », il doit répondre sur le clavier par « O » (c.-à-d. oui), ou « N » (non).

- Si la réponse est oui, l'ordinateur affiche « voici votre café »
- Dans le cas contraire il affiche « à bientôt ».

La première idée consiste à faire un teste conditionnel **Si** sur une variable de type caractère qui représente la réponse de l'utilisateur sur la question, s'il répond par 'O' c.-à-d. oui, sinon il veut dire non, mais la question qui se pose ici, est ce que dans les réponses différant de 'O' signifient non, **il se peut que c'est une faute de frappe, il vaut mieux qu'on prend ça en considération.** En affichant un message d'erreur, et on donne une autre chance à l'utilisateur pour ressaisir sa réponse jusqu'à une réponse par 'O' ou 'N' sera la meilleur solution qui est la suivante :

```
Algorithme distributeur
rep: caractère;
Début
Écrire ('voulez vous un café?');
Lire (rep);
Tant que ((rep ≠'O') et (rep ≠'N')) faire
 Écrire ('saisie erronée');
Lire (rep);
Fin tant que
si (rep ='O') alors
Écrire ('voici votre café');
sinon //rep='N'
   Écrire('à bientôt');
fin si
Fin.
```

2.2. La boucle « pour » ou « For » en C++:

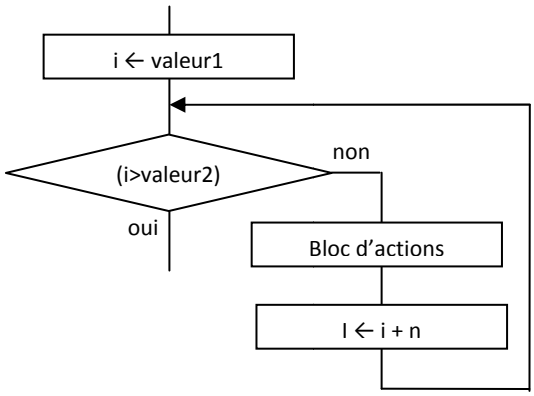
- La boucle pour permet de répéter une instruction ou un bloc d'instructions un nombre donné de fois.
- Elle utilise une variable appelée variable de contrôle, ou encore ***compteur*** d'itérations pour contrôler le nombre d'itérations.

Syntaxe :

Pour i **allant de** Valeur1 **à** Valeur2 (**pas**= n) **faire**

<Bloc d'action>

Fin Pour



Remarques :

- Les mots soulignés sont des mots clés
- Le schéma en face représente L'organigramme de la boucle Pour

Exemple :

Nous reprenons l'exemple précédent ; la somme des n premier nombres entiers positifs ; en utilisant maintenant la boucle **Pour**.

Algorithme som-ent

n , i , som :entier;

Début

Lire(n) ;

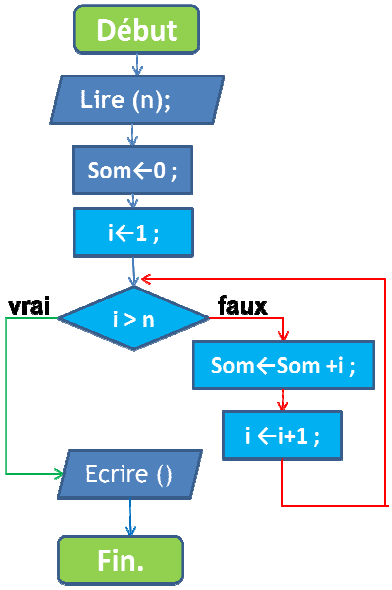
Som ← 0 ;

Pour i **allant de** 1 **à** n (pas = 1) **faire**

Som ← Som + i ;
Finpour
Ecrire (som) ;
Fin.

Remarques :

- Cet algorithme va s'arrêter des que **i atteint la valeur de n**. donc **à la sortie** de la boucle la valeur de **i = n + 1**.
- L'action **pour** est utilisée quand le nombre de fois **connu d'avance**. Pour cela, il faut préciser la valeur initiale et la valeur finale et éventuellement le pas.
- L'incrémentation du compteur se fait de manière automatique. Le pas, lorsqu'il n'est pas spécifié, est égal à 1.



2.2.2 Exercice : (Mot de passe d'un appareil téléphonique)

On vous demande d'écrire un algorithme qui affiche à l'utilisateur le message suivant: « entrez le mot de passe ». L'utilisateur saisie le mot de passe, s'il est correcte il affiche le message: « appareil déverrouillé » sinon, il affiche: « erreur, vous avez **trois** chances».

On peut le résoudre en utilisant la boucle pour, par ce que on sait à l'avance le nombre maximum de répétition en cas d'erreur de saisie de mot de passe (qui est 3 fois). La solution sera comme suit :

- on va afficher le message qui demande à l'utilisateur de saisir le mot de passe.
- Lire le mot de passe par une variable de type Chaine de caractères (pass).
- On va boucle 3 fois par la boucle pour (valeur1=3, valeur2=1, pas= -1), et chaque fois On fait un teste sur la variable pass, si elle est Différente de la valeur enregistrée dans l'appareil ('12345'), on affiche un message d'erreur et on donne à l'utilisateur la possibilité de ressaisir le mot de pass,
- à la sortie de la boucle pour, on fait un teste Si-sinon sur la valeur de pass.

```
Algorithme mot_de_passe
pass: chaine de caractère;
Début
Écrire ('entrez le mot de passe?');
Lire (pass);
Pour i allant de 3 à 1 (pas = -1)faire
Si (pass ≠'12345' )alors
Ecrire ('erreur, il vous reste',i,' chance');
Lire (pass);
Fin si
Fin pour
si (pass='12345') alors
Écrire ('appareil déverrouillé');
sinon //pass ≠'12345'
Écrire('ne pouvez pas utiliser l'appareil');
fin si
Fin.
```

Problème : dans le cas où l'utilisateur a saisie le mot de Passe correcte à la première fois, l'ordinateur va faire 3 itérations dans la boucle pour sans faire quelque chose !!! Puisque la condition de si est toujours fausse.

Pour résoudre ce problème, on doit utiliser une autre structure de boucle qui est la boucle **répéter** qu'on va l'étudier dans le prochain titre.

2.3. La boucle « Répéter » :

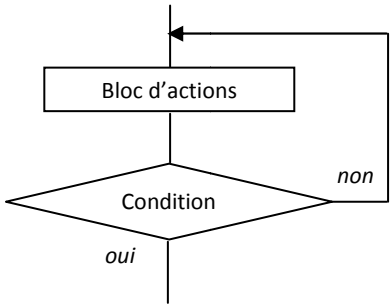
- Cette action exprime la répétition d'une ou plusieurs actions jusqu'à ce que la condition devienne vérifiée.

Syntaxe :

Répéter

<Bloc d'actions>

Jusqu'à (condition)

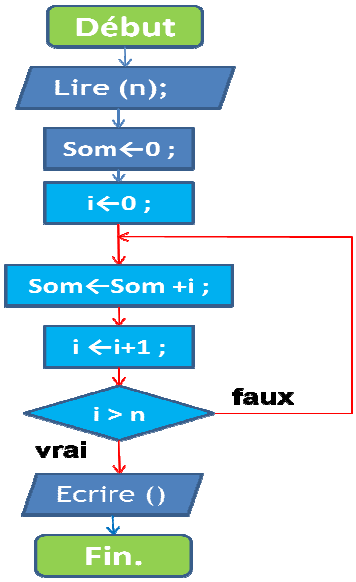


- Il s'agit de répéter l'exécution des instructions du bloc **<Bloc d'actions>** tant que la condition **n'est pas vérifiée** et arrêter leur exécution dès que la condition **devient vérifiée**.

Exemple :

Reprendre l'algorithme qui calcule la somme des N premiers nombres entiers positifs en utilisant l'action Répéter.

Algorithme som-ent4
n , i ,som:entier;
Début
Lire(n) ;
Som← 0 ; // la somme est initialisée à 0
i← 0 ; / / l'initialisation de i
Répéter
Som← Som + i ;
i←i + 1; // incrémentation de i
Jusqu'à (i> n)
Ecrire(som) ;
Fin.



Remarque :

- L'action « ***Répéter*** » est utilisée quand on ne connaît pas à priori le nombre de répétitions et lorsque **on doit exécuter les actions du bloc au moins une fois** (car l'évaluation de la condition se trouve à la fin).

2.3.1 Exercice : (Mot de passe d'un appareil téléphonique)

On va reprend l'exercice de déverrouillage d'un téléphone portable, on va le résoudre en utilisant la boucle répéter la solution sera comme suit :

- On met l'instruction qui permet d'afficher le message 'entrez le mot de passe', et initialise la variable i par le nombre maximum de chances qu'on va donner à l'utilisateur pour saisir le mot de passe ; i← 3 ;

```
Algorithme mot_de_passe
pass: chaine de caractère;
Début
Écrire ('entrez le mot de passe?');
```

- Puisque le principe de la boucle répéter est de faire exécuter le bloc d'instructions avant de faire évaluer la condition, on peut mettre l'instruction lire (pass), une seulle fois à l'intérieure de la boucle, puis on fait un test Si-finsi sur la valeur de pass
- La condition qui nous permet de sortir de la boucle répéter est : (pass='12345') ou (i < 1) ; c.-à-d. soit le mot de passe saisie est correcte ou le nombre de chances est épuisé (dans notre cas est 3) . donc l'algorithme est le suivant :

```
i← 3;
Répéter
Lire (pass);
Si (pass ≠'12345') alors
Ecrire ('erreur, vous avez ',i,' chance');
Fin si
i← i-1;
Jusqu'à ((pass='12345') ou (i<1))
si (pass='12345') alors
Écrire ('appareil déverrouillé');
sinon //pass ≠'12345'
Écrire('ne pouvez pas utiliser l'appareil');
fin si
Fin.
```

2.4. Structures de boucles imbriquées

On peut utiliser deux boucles ou plus, une à l'intérieur de l'autre, ils peuvent être des **pour** ou des **tant que** ou **répéter,** ou même un mélange entre eux. La structure sera comme suit

Tant que (condition 1) faire	Pour i allant de V1 à V2 pas =n faire	Répéter
Tant que (condition 2) faire	Pour j allant de V3 à V4 pas= m faire	Tant que (condition1) faire
<bloc d'instruction1>	<bloc d'instruction1>	<bloc d'instruction>
Fin tant que	Fin pour	Fin tant que
Fin tant que	Fin pour	Jusqu'à (condition 2)

Exemples :

Ecrire un algorithme qui lie un nombre naturel ***N*** calcule et affiche la somme :

$S = (1 + 2 + 3 + \cdots + N) + (2 + 3 + \cdots + N) + \cdots + (N-1 + N) + N.$

Algorithme S2
i, j, S, S′ : Entiers ;
Début
S ← 0
Pour i=1 `aN**Faire**
S′ ← 0
  Pour j=i `a N **Faire**
    S′ ← S′ + i
  Fin Pour
  S ← S + S′
Fin Pour
  Ecrire(S) ;
Fin.