Chapitre 3: Les structures itératives

Babacar DIOP

Adresse Mail: diopbabacar4888@gmail.com

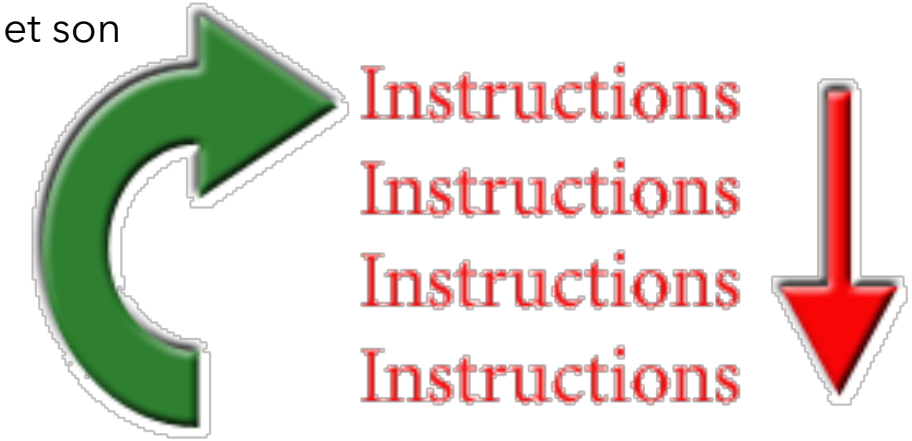
Téléphone: 777812040

Introduction

- Les structures itératives sont aussi appelées structures répétitives ou boucles.

- Elles permettent de répéter l'exécution d'un ou de plusieurs traitements jusqu'à ce qu'une certaine condition soit vérifiée.

- Les boucles sont classées en 3 catégories et chacune d'elles a sa syntaxe et son contexte d'utilisation. Les boucles sont :
  - la boucle **for**
  - la boucle **while**
  - la boucle **do-while**



La Boucle for

- **Contexte d'utilisation**: Elle ne peut être utilisée que si le nombre d'itérations est connu c'est-à-dire le nombre de fois que les actions de la boucle seront exécutées.

- **Syntaxe**:

```
for (initialisation ; condition ; mise à jour)
{
        <Action 1>
            ...
        <Action n>
}
```

- **Remarque**:Dans la boucle for, une variable de boucle appelée **indice de parcours** est utilisée pour contrôler la boucle. Cette variable est initialisée sur une valeur (**initialisation**), puis la boucle vérifie si cette variable est inférieure ou supérieure à la valeur du compteur (**condition**). Si l'instruction est vraie, le corps de la boucle est exécuté et sa variable est mise à jour (**mise à jour**). Les étapes sont répétées jusqu'à ce que la condition de sortie soit vérifiée.

Exercices d’application

- Exercice d’application 1: Ecrire un programme qui demande à l’utilisateur de saisir une valeur entière positive N. le programme affiche les nombres pairs compris entre 1 et N ainsi que la somme de ces nombres pairs.

- Exercice d’application 2: Ecrire un programme qui demande à l’utilisateur de saisir un nombre entier positif. Le programme détermine et affiche la table de multiplication de l’entier saisi. La limite de la table est fixée à 12.

La Boucle while

- **Contexte d'utilisation**: Dans la boucle for, nous avons constaté que le nombre d'itérations était connu à l'avance, c'est-à-dire que nous savons combien de fois le corps de la boucle doit être exécuté. La boucles while est utilisée dans des situations où nous ne connaissons pas le nombre exact d'itérations de boucle auparavant. L'exécution de la boucle est terminée sur la base d'une condition de sortie.
- **Syntaxe**:

```
//initialisation de l'indice de parcours
while (condition) {
        <Action 1>
            ...
        <Action n>

        //Mise à jour de l'indice
}
```

- **Remarque**: Parmi les actions de la boucle while, il en faut une qui permet la boucle de pouvoir évoluer.
- **Exercice d'application**: Ecrire un programme qui permet de saisir une valeur entière positive. Le programme détermine et affiche le factoriel de la valeur saisie.

La Boucle do..while

- **Contexte d'utilisation**: L'exécution de la boucle est également terminée sur la base d'une condition de sortie. La **différence principale** entre la boucle do..while et la boucle while est que dans la boucle do..while teste à la fin du corps de la boucle;

- **Syntaxe**:

```
// initialisation indice de parcours
do{
        <Action 1>
            ...
        <Action n>

        //mise à jour indice
} while(condition);
```

- **Remarque**: il existe un point-virgule (;) à la fin de la boucle.

Exercices d'application

- Exercice d'application 1: Ecrire un programme qui demande à l'utilisateur de saisir un nombre entier positif. Le programme détermine et affiche si le nombre saisi est un nombre premier ou pas. Un nombre est premier s'il n'a que deux diviseurs qui sont 1 et le nombre lui-même.

- Exercice d'application 2: Ecrire un programme qui demande à l'utilisateur de saisir une valeur entière positive n. le programme calcule et affiche le résultat de l'expression ci-dessous :

$$\sum_{i=1}^{n} [n - 2 * i]$$

Imbrication des structures répétitives

- **Contexte d'utilisation**: Dans un programme, il est fréquent qu'une boucle soit définie à l'intérieur d'une autre boucle alors on parle d'imbrication de boucles. Dans ce cas, la boucle la plus interne est fermée avant celle la plus externe.

- Exercice d'application 1: Ecrire un programme qui demande à l'utilisateur de saisir une valeur entière positive n. le programme calcule et affiche le résultat de l'expression ci-dessous :

$$\sum_{i=1}^{n} \left[(i!) + (n)^i - ((3 * i)/n) \right]$$

- Exercice d'application 2: Ecrire un programme qui permet de contrôler la saisie d'une valeur entière positive N. Le programme détermine et affiche les tables de multiplication des nombres pairs compris entre 1 et N. La limite de chaque table est fixée à 12.

- Exercice d'application 3: Ecrire un programme qui permet de saisir une série de N valeurs entières positives. Le programme détermine et affiche la moyenne des nombres pairs ainsi que le pourcentage de présence des nombres impairs divisibles par 5.

  Moyenne des nombres pairs = Somme des nombres pairs / Nombre de nombres pairs
  Pourcentage de présence des nombres impairs = (Nombre de nombres impairs * 100) / N