-Les Structures de contrôle-

https://goo.gl/J4R515

I. INTRODUCTION

Introduction

Les instructions d'un Algorithme sont exécutées dans l'ordre les unes après les autres.

```
Instruction 1 ;
Instruction 2 ;
Instruction 3 ;
Instruction 4 ;
Instruction 5 ;
…
Instruction N ;
```

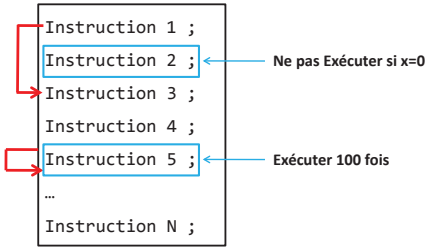

I. Introduction

Dans Certains Cas, Nous avons besoin de modifier l'ordre d'exécution des instructions :

Par exemple :

→Ne Pas Exécuter l' Instruction 2 si la variable **x** vaut **0**

→Exécuter L' Instruction 5 **100** fois



I. Introduction

Les **Structures de Contrôle** permettent de contrôler l'ordre d'exécution des instructions (le séquencement) d'un algorithme.

Il existe deux types de structures de Contrôle fondamentales :

- **Les Structures Conditionnelles (Condition) :** Permettent d'exécuter une suite d'instructions (bloc) si une condition est vérifiée (vrai).
- **Les Structures Itératives (Boucle, Répétition) :** Permettent de répéter l'exécution d'une suite d'instructions (bloc) un certain nombre de fois.

II. Les Structures Conditionnelles

II. Les Structures Conditionnelles

En programmation C++ il existe trois (03) types de structures conditionnelles:

- **La Condition à un Choix (if)**
- **La Condition à deux Choix (if…else)**
- **La Condition à choix multiple (switch)**

II. Les Structures Conditionnelles

• La Condition à Un Choix.

Syntaxe :

```
Si (Condition) Alors
 Bloc d'Instructions;
FinSi
```

En C++:

```
if (Condition) Instruction1;

if (Condition) {Bloc d'Instructions;}
```

Condition : Expression Booléenne (Vrai ou Faux).
Bloc D'Instructions : Suite d'instructions (une ou plusieurs instructions).

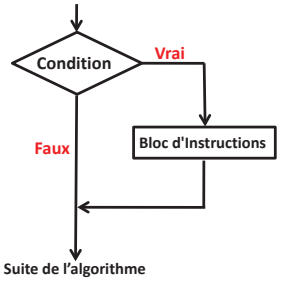
Explication :

Bloc d'Instructions est exécuté seulement si la **Condition** est vraie.

II. Les Structures Conditionnelles

• La Condition à Un Choix.

Organigramme :



II. Les Structures Conditionnelles

• La Condition à Un Choix.

Exemple 1 :
Ecrire un algorithme qui demande à l'utilisateur un Nombre entier, puis affiche le message "**Le Nombre est Nul**" si le Nombre vaut **0**

```
Algo Zero
Var N : Entier;
Début
Ecrire("Donner un Nombre :");
Lire(N);
Si (N=0) Alors
 Ecrire("N est nul");
FinSi
FIN.
```


II. Les Structures Conditionnelles

• La Condition à Un Choix.

Exemple 2 :
Ecrire un algorithme qui demande à l'utilisateur un Nombre réel, puis affiche le message " **Ce Nombre est Positif** " si le Nombre est **> 0**

```
Algo Positif
Var r : réel;
Début
Ecrire("Donner un Nombre :");
Lire(r);
Si (r>0) Alors
 Ecrire("Ce Nombre est positif");
FinSi
FIN.
```


II. Les Structures Conditionnelles

• La Condition à Deux Choix.

Syntaxe :

```
Si (Condition) Alors
 Bloc d'Instructions1;
Sinon
 Bloc d'Instructions2;
FinSi
```

En C++:

```
if (Condition) Instruction1;
else  Instruction2;
```

```
if (Condition) {Bloc d'Instructions1;}
else {Bloc d'Instructions2;}
```

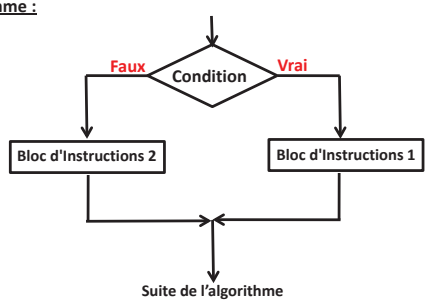
Explication :

Si la condition est vraie **Bloc d'Instructions 1** sera exécuté
Sinon **Bloc d'Instructions 2** sera exécuté

II. Les Structures Conditionnelles

• La Condition à Deux Choix.

Organigramme :



II. Les Structures Conditionnelles

• La Condition à Deux Choix.

Exemple 3 :

Ecrire un algorithme qui demande à l'utilisateur un Nombre entier **N** puis affiche un message indiquant si le Nombre est **Pair** ou **Impair**.

```
Algo Pair
Var
N : Entier;
Début
 écrire("Donner un Nombre");
 lire(N);
 Si (N mod 2=0) Alors
  écrire("N est Pair");
 Sinon
  écrire("N est Impair");
 FinSi
FIN.
```


II. Les Structures Conditionnelles

• La Condition à Deux Choix.

Exemple 4 : (Imbrication)
Ecrire un algorithme qui demande à l'utilisateur un Nombre entier **N** puis affiche un message indiquant si le Nombre est Strictement **Positif**, Strictement **Négatif** ou **Nul**.

```
Algo Nbre
Var
N : Entier;
Début
Ecrire("Donner un Nombre");
Lire(N);
Si (N>0) Alors
     Ecrire("N est Strictement Positif");
Sinon
   Si (N=0) Alors
     Ecrire("N est Nul");
   Sinon
     Ecrire("N est Strictement Négatif");
   FinSi
FinSi
FIN.
```


II. Les Structures Conditionnelles

• La Condition à Deux Choix.

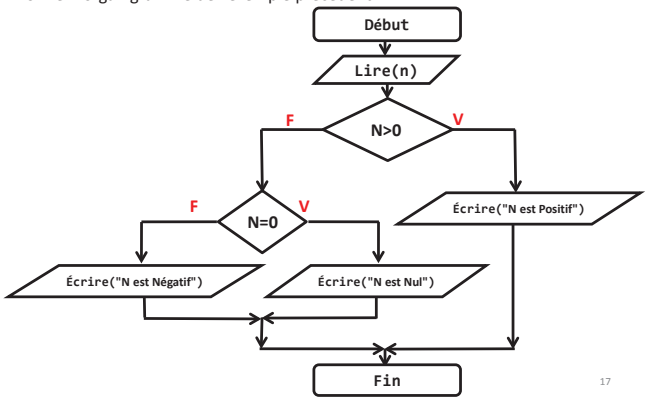
Exemple 4 : (Imbrication)
Ecrire un algorithme qui demande à l'utilisateur un Nombre entier **N** puis affiche un message indiquant si le Nombre est Strictement **Positif**, Strictement **Négatif** ou **Nul**.

```
Algo Nbre
Var
N : Entier;
Début
Ecrire("Donner un Nombre");
Lire(N);
Si (N>0) Alors
     Ecrire("N est Strictement Positif");
SinonSi (N=0) Alors
     Ecrire("N est Nul");
Sinon
     Ecrire("N est Strictement Négatif");
FinSi
FIN.
```


II. Les Structures Conditionnelles

• La Condition à Deux Choix.

Donner l'organigramme de l'exemple précédent



II. Les Structures Conditionnelles

• La Condition à Choix Multiple :

Syntaxe	En C++

Selon (VE) vaut
Val1:Bloc1;
Val2:Bloc2;
...
ValN:BlocN;
autre:BlocM;
FinSelon
	switch(VE) {
	case Val1:Bloc1; break;
	case Val2:Bloc2; break;
	...
	case ValN:BlocN; break;
	default:BlocM;
	}

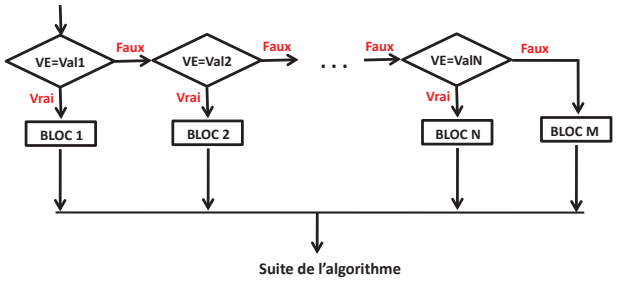
VE : Variable ou Expression.

Si **VE** n'a aucune valeur parmi (**Val1**, **Val2**,....., **ValN**) , le bloc **autre** sera exécuté.

II. Les Structures Conditionnelles

• La Condition à Choix Multiple :

Organigramme :



II. Les Structures Conditionnelles

• La Condition à Choix Multiple :

Exemple 5 :

Ecrire un algorithme qui demande à l'utilisateur un Nombre compris entre 0 et 3 puis affiche ce Nombre en lettre.

```
Algo Nombre
Var
N : Entier;
Début
écrire("Donner un Nombre entre 0 et 3 :");
Lire(N);
Selon (N) vaut
 0 : Ecrire("Zéro");
 1 : Ecrire("Un");
 2 : Ecrire("Deux");
 3 : Ecrire("Trois");
 autre : Ecrire("Erreur");
FinSelon
FIN.
```


III. Les Structures Itératives

III. Les Structures Itératives

Les Structures Itératives (Boucles, Répétitions) permettent d'exécuter un bloc d'instructions plusieurs fois.
Le Nombre de répétitions peut être connu ou dépendre d'une condition.

En programmation C++ il existe trois (03) types de structures itératives:

- La Boucle **Tantque … Faire** **(while)**
- La Boucle **Faire… Tantque** **(do-while)**
- La Boucle **Pour… Faire** **(for)**

III. Les Structures Itératives

1. La Boucle **Tantque … Faire… FinTantque**

Syntaxe :

```
Tantque (Condition) Faire
 Bloc d'Instructions;
FinTantque
```

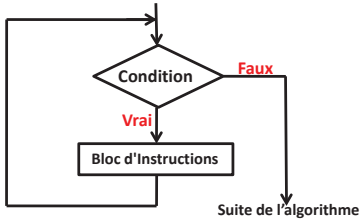
En C++:

```
while (Condition) {
  Bloc d'Instructions;
}
```

Explication :

Bloc d'instructions est répété tant que la condition est vraie.

Organigramme:



III. Les Structures Itératives

1. La Boucle **Tantque … Faire… FinTantque**

Exemple 6

Ecrire un algorithme qui lit un Nombre entier N puis affiche tous les Nombres de 0 à N.

```
Algo Nombre
Var
N,i : Entier;
Début
Ecrire("Donner un Nombre N");
Lire(N);
i←0;
Tantque (i<=N) Faire
    Ecrire(i);
    i ← i+1;
FinTantque
FIN.
```


III. Les Structures Itératives

1. La Boucle **Tantque … Faire… FinTantque**

Exemple 7

Ecrire un algorithme qui lit un Nombre entier N puis affiche son double.
L'algorithme doit être répéter et s'arrêter lorsque le Nombre vaut **-1**.

```
Algo Nombre
Var
N,i : Entier;
Début
Ecrire("Donner N:");
Lire(N);
Tantque (N<>-1) Faire
    Ecrire("Le double de N est:",2*N);
    Ecrire("Donner N:");
    Lire(N);
FinTantque
FIN.
```


III. Les Structures Itératives

2. La Boucle **Faire… Tantque**

Syntaxe

```
Faire
 Bloc d'Instructions;
Tantque (Condition);
```

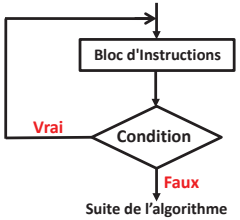
En C++:

```
do {
   Bloc d'Instructions;
   } while (Condition);
```

Explication :

Le Bloc d'instructions sera au moins exécuté **une seule fois**.
L'exécution du Bloc d'instructions est répétée Tant que la Condition est vraie.

Organigramme :



III. Les Structures Itératives

2. La Boucle **Faire… Tantque**

Exemple 8 :

Refaire l'Exemple 7 en utilisant la structure **Faire…Tantque:**

Ecrire un algorithme qui lit un Nombre entier N puis affiche tous les Nombres de 0 à N.

```
Algo Nombre2
Var
N,i : Entier;
Début
Ecrire("Donner un Nombre N");
Lire(N);
i←0;
Faire
    Ecrire(i);
    i ← i+1;
Tantque (i<=N);
FIN.
```

III. Les Structures Itératives

Différence entre **Tantque…Faire** et **Faire…Tantque**

Question

Quelle est la différence entre les deux boucles ?

Tantque *(Condition)* **Faire** *Bloc d'instructions;* **FinTantque**	**Faire** *Bloc d'instructions;* **Tantque** *(Condition);*

→Dans la boucle **Tantque…Faire** la condition est d'abord testée puis le bloc d'instruction est exécutée si la condition est vrai. Si la condition est initialement fausse le bloc d'instruction ne sera pas exécuté.

→Dans la boucle **Faire…Tantque**, le bloc d'instruction est d'abord exécuté puis la condition est testée. Le bloc d'instruction est exécuté au moins une fois.

III. Les Structures Itératives

Différence entre **Tantque…Faire** et **Faire…Tantque**

Exemple 9

Ecrire un algorithme qui lit un nombre N puis affiche tous les Nombres pairs de 0 à N.

```
Algo Pair1
Var
N,i: Entier;
Début
Ecrire("Donner N:");
Lire(N);
i←0;
Tantque (i<=N) Faire
   Ecrire(i);
   i ← i+2;
FinTantque
FIN.
```

```
Algo Pair2
Var
N,i : Entier;
Début
Ecrire("Donner N:");
Lire(N);
i←0;
Faire
   Ecrire(i);
   i ← i+2;
Tantque (i<=N);
FIN.
```


III. Les Structures Itératives

Différence entre **Tantque…Faire** et **Faire…Tantque**

Exemple

```
X←0;
Tantque (x<>0) Faire
    Ecrire("Bonjour");
FinTantque
```

Le message "Bonjour" ne sera pas affiché car la condition est fausse

```
X←0;
Faire
  Ecrire("Bonjour");
Tantque (x<>0);
```

Le message "Bonjour" sera affiché une seule fois

III. Les Structures Itératives

3. La Boucle **Pour … Faire …FinPour** :

Syntaxe

```
Pour IdVar de V0 à Vf Faire
 Bloc d'instructions
FinPour
```

En C++:

```
for(IdVar=V0;IdVar<=Vf;IdVar++)
 {
  Bloc d'Instructions;
 }
```

Explication :

IdVar : Variable appelée **Compteur**.
V0 : Valeur Initiale du Compteur. Peut être une Constante, Une Variable ou une expression.
Vf : Valeur Finale du Compteur. Peut être une Constante, Une Variable ou une expression.

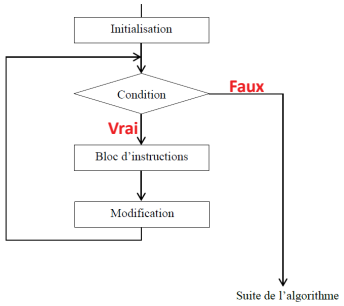
Le bloc d'instruction est répété Tant que ce que le compteur atteigne la Valeur finale **Vf**.
À Chaque itération (répétition), la valeur du Compteur (**IdVar**) évolue selon la valeur du pas **1** :

IdVar ← IdVar + 1

III. Les Structures Itératives

3. La Boucle **Pour … Faire …FinPour** :

Organigramme :



III. Les Structures Itératives

3. La Boucle **Pour** :

Exemple 10 :
Reprendre l'Exemple 7 en utilisant une boucle **Pour.**
Ecrire un algorithme qui lit un Nombre entier N, puis affiche tous les Nombres de 0 à N.

```
Algo Nombre
Var
N,i : Entier;
Début
Ecrire("Donner N");
Lire(N);
i←0;
Tantque (i<=N) Faire
    Ecrire(i);
    i ← i+1;
FinTantque
FIN.
```

```
Algo Nombre2
Var
N,i : Entier;
Début
Ecrire("Donner N");
Lire(N);
i←0;
Faire
    Ecrire(i);
    i ← i+1;
Tantque (i<=N);
FIN.
```

```
Algo Nombre3
Var
N,i : Entier;
Début
Ecrire("Donner N");
Lire(N);
Pour i de 0 à N Faire
 Ecrire(i);
FinPour
FIN.
```

Le compteur de la boucle **Pour** est initialisé automatiquement et est incrémenté automatiquement

Choix de la structure répétitive

Dans ce chapitre, nous avons vu trois façons différentes de programmer des boucles (**while**, **do-while**, **for**). Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser, en respectant toutefois les directives suivantes :

* Si le bloc d'instructions ne doit pas être exécuté si la condition est fausse, alors utilisez **while** ou **for**

* Si le bloc d'instructions doit être exécuté au moins une fois, alors utilisez **do-while**

* Si le nombre d'exécutions du bloc d'instructions dépend d'une ou de plusieurs variables qui sont modifiées à la fin de chaque répétition, alors utilisez **for**

* Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie (aussi longtemps qu'il y a des données dans le fichier d'entrée), alors utilisez **while**

Le choix entre **for** et **while** n'est souvent qu'une question de préférence ou d'habitudes:

* **for** nous permet de réunir avantageusement les instructions qui influencent le nombre de répétitions au début de la structure.

* **for** a le désavantage de favoriser la programmation de structures surchargées et par la suite illisibles.

* **while** a l'avantage de correspondre plus exactement aux structures d'autres langages de programmation (while, tant que).

* **while** a le désavantage de mener parfois à de longues structures, dans lesquelles il faut chercher pour trouver les instructions qui influencent la condition de répétition.