Structures de contrôle répétitives

Les structures répétitives nous permettent de **répéter un traitement un nombre fini de fois**.

Exemple : afficher tous les nombres premiers entre 1 et N (N nombre entier positif donné).

Nous avons **trois types** de structures itératives (boucles) :

1 *La boucle*

Pour

Le nombre de répétitions est connu à l'avance.

2 *La boucle*

Tantque

On répète tant que la condition est vraie (testée avant).

3 *La boucle*

Répéter

On répète jusqu'à ce que la condition soit vraie (testée après).

Boucle Tant-que (While)

La structure de contrôle répétitive **tant-que** (**while** en langage C) utilise une **expression logique** ou booléenne comme **condition d'accès** à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on **entre** à la boucle, sinon on la **quitte**.

La syntaxe de la boucle **tant-que** est comme suit :

Algorithme

```
tant-que <condition> faire
    <instruction(s)> ;
fin tant-que;
```

Langage C

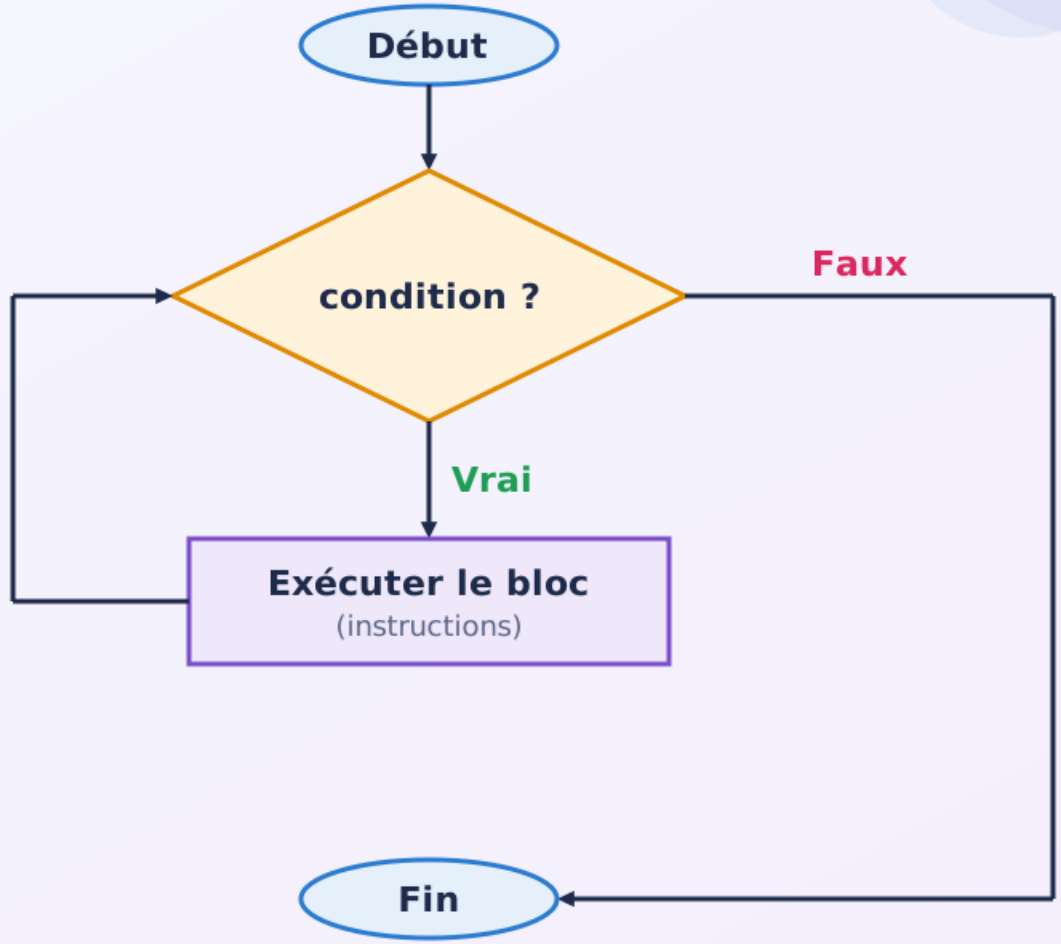
```
while (<condition>)
{
    <instruction(s)>;
}
```

`<condition>` : expression logique qui peut être **vraie** ou **fausse**.

Organigramme de la boucle Tant-que

Principe

1. Test de la condition : `condition ?`
2. Vrai : exécuter le bloc, puis retour au test
3. Faux : sortie de la boucle et passage à Fin
4. Le test est fait **avant** : le bloc peut ne jamais s'exécuter.



Boucle Tant-que (While)

Valeur initiale + incrémentation

On ajoute la **valeur initiale** et l'**incrémentation** du compteur dans l'algorithme, le programme C et l'organigramme.

Algorithme

```
cpt ← Vi ;
tant-que cpt <= Vf faire
    <instruction(s)> ;
    cpt ← cpt + 1 ;
fin tant-que;
```

1 Valeur initiale

Le compteur **cpt** reçoit sa valeur initiale **Vi avant** d'entrer dans la boucle.

2 Incrémentation

Le compteur est modifié à **chaque tour** jusqu'à dépasser la valeur finale **Vf**.

Langage C

```
cpt = Vi;
while (cpt <= Vf)
{
    <instruction(s)>;
    cpt++;
}
```

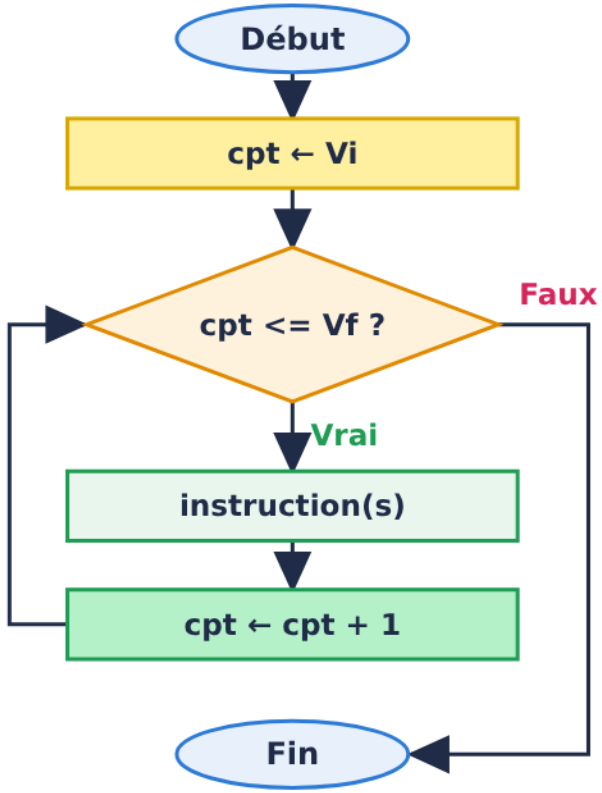
1 Valeur initiale

Affectation du compteur **avant** le **while**, par exemple **cpt = 1;** (**Vi** = valeur initiale)

2 Incrémentation

À la fin du bloc : **cpt++;** ou **cpt = cpt + 1;**
Sans elle, la boucle est infinie.

Organigramme



Somme des N premiers entiers

Énoncé

Boucle **Tant que** : calculer la somme suivante

Calculer la **somme** des **N** premiers entiers :

$$S = 1 + 2 + 3 + \cdots + N$$

L'entier **N** est saisi par l'utilisateur ; le résultat **S** est affiché à la fin.

1 Algorithme

Écrire l'algorithme en utilisant une boucle `Tant que` avec les variables `N` `i` `S`.

2 Langage C

Traduire l'algorithme en un **programme en langage C** avec `scanf` `while` `printf`.

3 Organigramme

Dessiner l'**organigramme** : début, saisie de N, boucle `i ≤ N ?`, affichage, fin.

Exemple

Somme des N premiers entiers

Boucle tant-que : calculer S = 1 + 2 + 3 + ... + N

Algorithme

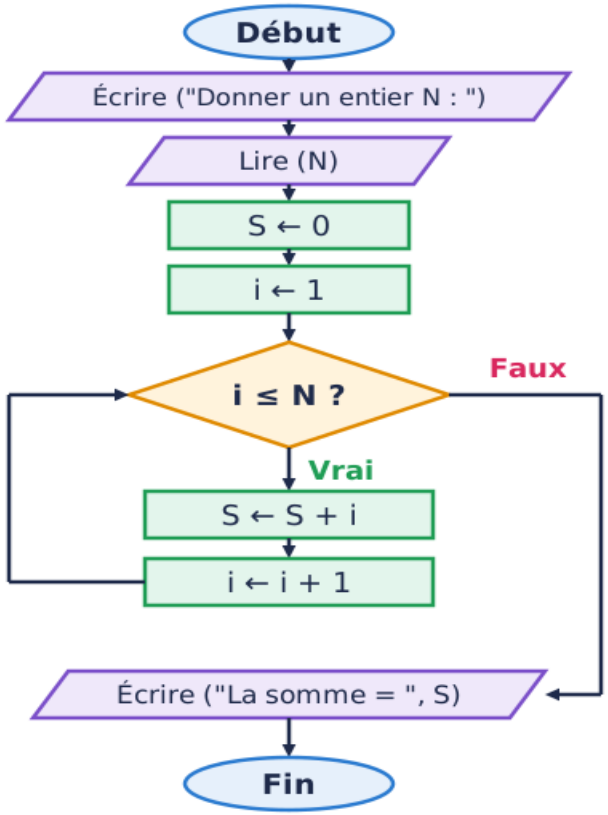
```
Algorithme Somme ;
Variables
    N, i, S : entier ;
Début
    Écrire ("Donner un entier N : ") ;
    Lire (N) ;
    S ← 0 ;
    i ← 1 ;
    tant-que i ≤ N faire
        S ← S + i ;
        i ← i + 1 ;
    fin tant-que;
    Écrire ("La somme = ", S) ;
Fin.
```

Langage C

```c
#include <stdio.h>

int main()
{
    int N, i, S;
    printf("Donner un entier N : ");
    scanf("%d", &N);
    S = 0;
    i = 1;
    while (i <= N)
    {
        S = S + i;
        i = i + 1;
    }
    printf("La somme = %d", S);
    return 0;
}
```

Organigramme



Remarque

Boucle pour et boucle tant-que

Il est possible de remplacer **toute** boucle « **pour** » par une boucle « **tant-que** », cependant, l'inverse **n'est pas toujours réalisable**.

pour ⟶ tant-que

✓ Toujours possible

```
pour i ← 1 à N faire
    instructions
finPour;
```

↓

```
i ← 1 ;
tant-que i ≤ N faire
    instructions
    i ← i + 1 ;
fin tant-que;
```

tant-que ⟶ pour

✗ Pas toujours réalisable

La boucle « **pour** » exige un nombre d'itérations **connu à l'avance** (de vi à vf).

La boucle « **tant-que** » dépend seulement d'une **condition** : le nombre de répétitions peut être inconnu.

Exemple : lire des entiers jusqu'à la saisie de 0

```
tant-que x ≠ 0 faire
    …
fin tant-que;
```

Structures de contrôle répétitives

Les structures répétitives nous permettent de **répéter un traitement un nombre fini de fois**.

Exemple : afficher tous les nombres premiers entre 1 et N (N nombre entier positif donné).

Nous avons **trois types** de structures itératives (boucles) :

1 *La boucle*

Pour

Le nombre de répétitions est connu à l'avance.

2 *La boucle*

Tantque

On répète tant que la condition est vraie (testée avant).

3 *La boucle*

Répéter

On répète jusqu'à ce que la condition soit vraie (testée après).

Boucle Répéter (do ... while)

La structure de contrôle répétitive **répéter** utilise une **expression logique** ou booléenne comme **condition de sortie** de la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) on **sort** de la boucle, sinon on y accède (on **répète** l'exécution du bloc).

La syntaxe de la boucle **répéter** est comme suit :

Algorithme

```
répéter
    <instruction(s)> ;
jusqu'à <condition>;
```

Langage C

```
do
{
    <instruction(s)>;
} while (<condition2>);
```

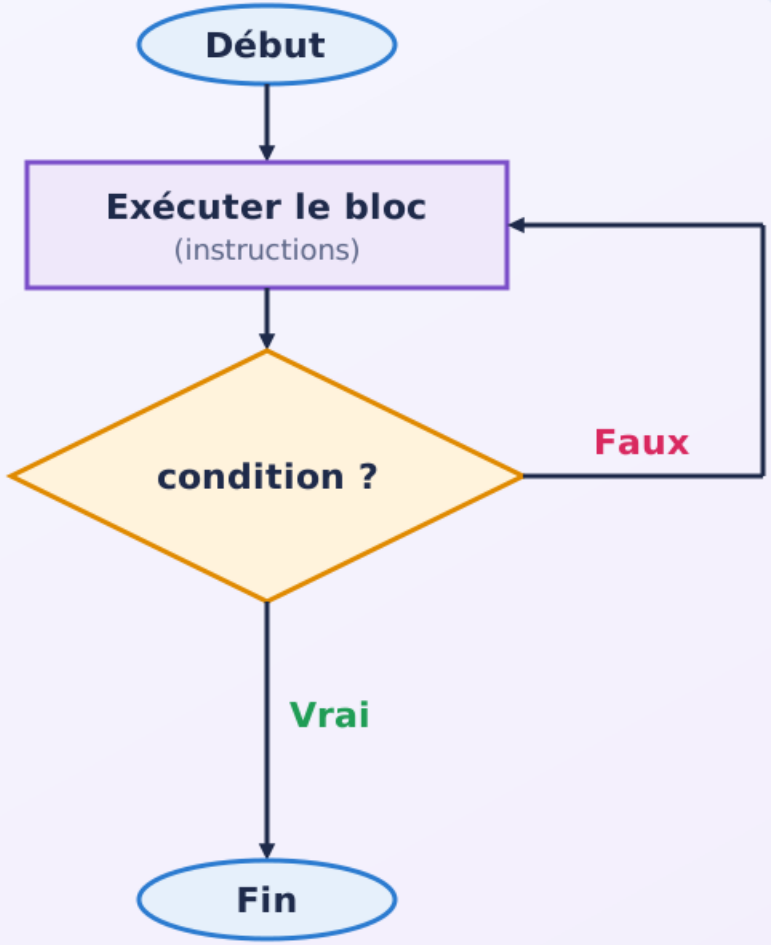
`<condition>` : expression logique qui peut être **vraie** ou **fausse**.

`<condition2>` : expression logique **inverse** de `<condition>`.

Organigramme de la boucle Répéter

Principe

1. Exécuter le bloc d'instructions (au moins une fois)
2. Test de la condition de sortie : `condition ?`
3. Vrai : sortie de la boucle et passage à Fin
4. Faux : retour au bloc, on répète l'exécution



Boucle Répéter (Do...While)

Valeur initiale + incrémentation

On ajoute la **valeur initiale** et l'**incrémentation** du compteur dans l'algorithme, le programme C et l'organigramme.

Algorithme

```
cpt ← Vi ;
répéter
    <instruction(s)> ;
    cpt ← cpt + 1 ;
jusqu'à cpt > Vf ;
```

1 Valeur initiale

Le compteur **cpt** reçoit sa valeur initiale **Vi avant** d'entrer dans la boucle.

2 Incrémentation

Le compteur est modifié à **chaque tour**. On s'arrête quand **cpt > Vf** (condition d'arrêt).

Langage C

```
cpt = Vi;
do
{
    <instruction(s)>;
    cpt++;
} while (cpt <= Vf);
```

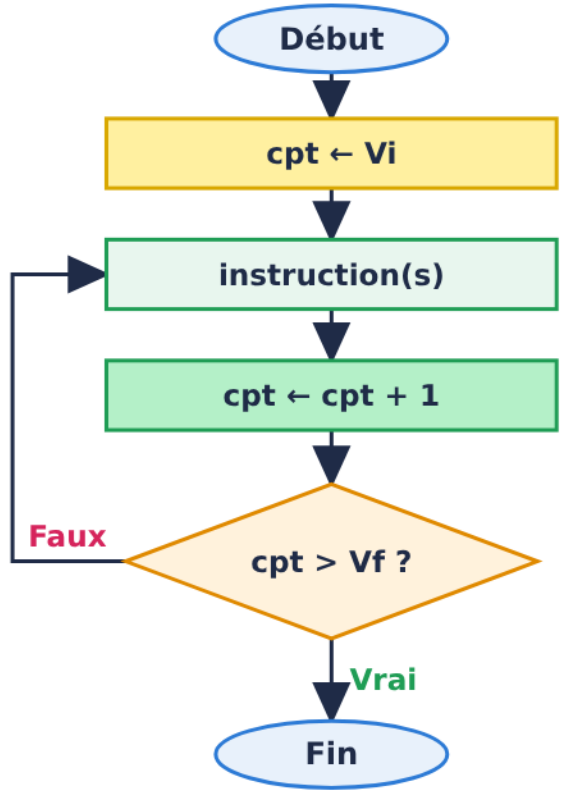
1 Valeur initiale

Affectation du compteur **avant** le **do**, par exemple **cpt = 1;** (**Vi** = valeur initiale)

2 Incrémentation

cpt++; dans le bloc, puis **while (cpt <= Vf)** : on continue tant que c'est vrai. Le bloc s'exécute **au moins 1 fois**.

Organigramme



Somme des N premiers entiers

Énoncé

Boucle **Répéter (Do...While)** : calculer la somme suivante

Calculer la **somme** des **N** premiers entiers :

$$S = 1 + 2 + 3 + \cdots + N$$

L'entier **N** est saisi par l'utilisateur ; le résultat **S** est affiché à la fin.

1 Algorithme

Écrire l'algorithme en utilisant une boucle `Répéter` avec les variables `N` `i` `S`.

2 Langage C

Traduire l'algorithme en un **programme en langage C** avec `scanf` `do…while` `printf`.

3 Organigramme

Dessiner l'**organigramme** : début, saisie de N, boucle, test d'arrêt `i > N ?`, affichage, fin.

Exemple

Somme des N premiers entiers

Boucle répéter : calculer S = 1 + 2 + 3 + ... + N

Algorithme

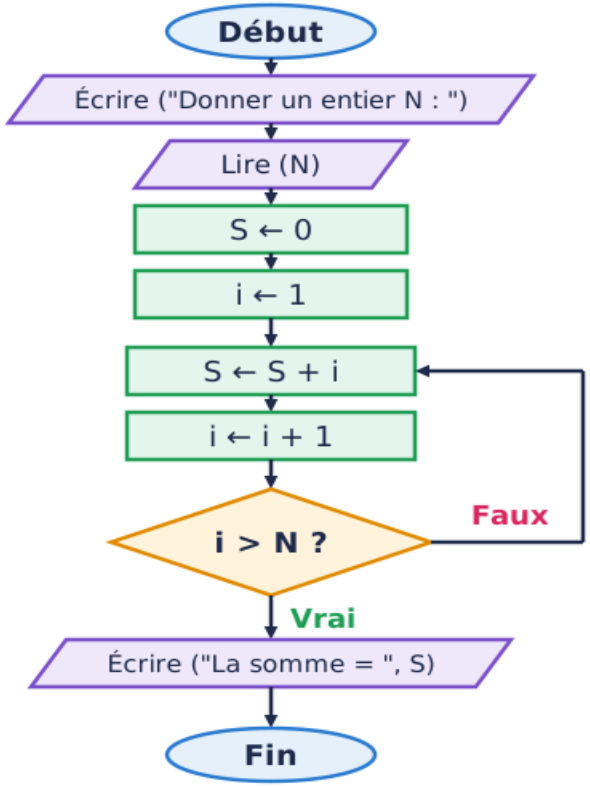
```
Algorithme Somme ;
Variables
    N, i, S : entier ;
Début
    Écrire ("Donner un entier N : ") ;
    Lire (N) ;
    S ← 0 ;
    i ← 1 ;
    répéter
        S ← S + i ;
        i ← i + 1 ;
    jusqu'à i > N ;
    Écrire ("La somme = ", S) ;
Fin.
```

Langage C

```c
#include <stdio.h>

int main()
{
    int N, i, S;
    printf("Donner un entier N : ");
    scanf("%d", &N);
    S = 0;
    i = 1;
    do
    {
        S = S + i;
        i = i + 1;
    } while (i <= N);
    printf("La somme = %d", S);
    return 0;
}
```

Organigramme



Exercice 02

Algorithme → Programme

Soit l'algorithme suivant : **Exo2**

A Algorithme Exo2

```
Algorithme Exo2 ;
Variables
    X, P, S : réel ;
    i, N : entier ;
Début

    // Entrées
    Ecrire ("Donner les valeurs de N et X : ") ;
    Lire (N, X) ;

    // Traitement
    S ← 0 ;
    P ← X ;
    Pour i ← 1 à N faire
        S ← S + P / i ;
        P ← P * X ;
    Fin-Pour

    // Sortie
    Ecrire ("S= ", S) ;
Fin.
```

1. Traduire l'algorithme en un **programme en langage C**.
2. Compiler et exécuter le programme pour `N=3` et `X=3`.
3. Dérouler le programme pour `N=3` et `X=3`.
4. Déduire l'**expression générale** du résultat `S` en fonction de `X` et `N` ?
5. Ré-écrire l'algorithme/programme en remplaçant la boucle **Pour** par la boucle **Tant-que**.
6. Ré-écrire l'algorithme/programme en remplaçant la boucle **Pour** par la boucle **Répéter**.
7. Modifier l'algorithme pour calculer la somme **S2** :

$$S2 = X + \frac{X^3}{2} + \frac{X^5}{3} + \frac{X^7}{4} + \cdots + N^{ème}\ \text{terme}$$

Corrigé de l'exercice 02

Algorithme → Langage C

5) Réécrire l'algorithme en remplaçant la boucle ***Pour*** par la boucle ***Tant-que***.

A Algorithme

```
Algorithme Exo2 ;
Variables
    X, P, S : réel ;
    i, N : entier ;
Début
    // Entrées
    Ecrire ("Donner les valeurs de N et X : ") ;
    Lire (N, X) ;

    // Traitement
    S ← 0 ;
    P ← X ;
    i ← 1 ;
    Tant-que i ≤ N faire
        S ← S + P / i ;
        P ← P * X ;
        i ← i + 1 ;
    Fin Tant-que

    // Sortie
    Ecrire ("S =", S:4:3) ;
Fin.
```

C Programme C

```c
#include<stdio.h>
int main()
{
    float X,P,S;
    int i,N;

    // Entrées
    printf("Donner les valeurs de N et X \n");
    scanf("%d %f", &N,&X);

    // Traitement
    S=0;
    P=X;
    i=1;
    while (i<=N)
    {
        S=S+P/i;
        P=P*X;
        i++;
    }

    // Sortie
    printf("S=%.3f",S);
    return 0;
}
```

Corrigé de l'exercice 02

Algorithme → Langage C

6) Réécrire l'algorithme en remplaçant la boucle ***Pour*** par la boucle ***Répéter***.

A Algorithme

```
Algorithme Exo2 ;
Variables
    X, P, S : réel ;
    i, N : entier ;
Début
    // Entrées
    Ecrire ("Donner les valeurs de N et X : ") ;
    Lire (N, X) ;

    // Traitement
    S ← 0 ;
    P ← X ;
    i ← 1 ;
    Répéter
        S ← S + P / i ;
        P ← P * X ;
        i ← i + 1 ;
    Jusqu'à i > N ;

    // Sortie
    Ecrire ("S =", S) ;
Fin.
```

C Programme C

```
#include<stdio.h>
int main()
{
    float X,P,S;
    int i,N;

    // Entrées
    printf("Donner les valeurs de N et X \n");
    scanf("%d %f", &N,&X);

    // Traitement
    S=0;
    P=X;
    i=1;
    do
    {
        S=S+P/i;
        P=P*X;
        i++;
    } while (i<=N);

    // Sortie
    printf("S=%.3f",S);
    return 0;
}
```

Notion d'algorithme et de programme

Structure de contrôle

Structure conditionnelle

Test alternatif simple

- Permet de décider si on exécute une ou plusieurs instructions ou non
- Contient un seul bloc d'instructions

Algorithme :

```
Si (condition) alors
    <Instruction(s) Si>;
Fin-Si;
```

Langage C :

```
if (condition)
{
    <Instruction(s) If>;
}
```

Tel que :

cpt : compteur (valeur entière)
Vi : valeur initiale
Vf : valeur finale

Test alternatif double

- Permet de décider si on exécute une séquence d'instructions ou une autre séquence d'instructions
- Contient deux blocs d'instructions

Algorithme :

```
Si (condition) alors
    <Instruction(s) Si>;
Sinon
    <Instruction(s) Sinon>;
Fin-Si;
```

Langage C :

```
if (condition)
{
    <Instruction(s) If>;
}
else
{
    <Instruction(s) Else>;
}
```

Test alternatif imbriqué

- Permet de choisir entre plusieurs séquences d'instructions selon plusieurs conditions successives
- Contient un test alternatif à l'intérieur d'un autre (Si dans Sinon)

Algorithme :

```
Si (condition1) alors
    <Instruction(s) 1>;
Sinon
    Si (condition2) alors
        <Instruction(s) 2>;
    Sinon
        <Instruction(s) 3>;
    Fin-Si;
Fin-Si;
```

Langage C :

```
if (condition1)
{
    <Instruction(s) 1>;
}
else
{
    if (condition2)
    {
        <Instruction(s) 2>;
    }
    else
    {
        <Instruction(s) 3>;
    }
}
```

Structure répétitive

Boucle Pour (For)

La structure de contrôle répétitive **Pour (for)** utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique.

Algorithme :

```
Pour cpt ← Vi à Vf faire
    <Instruction(s)>;
Fin-Pour;
```

Langage C :

```
for (cpt=Vi; cpt<=Vf; cpt++)
{
    <Instruction(s)>;
}
```

Boucle Tant-que (While)

La structure de contrôle répétitive **Tant-que (while)** utilise une expression logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée donc on entre à la boucle, sinon on la quitte.

Algorithme :

```
cpt ← Vi ;
Tant-que (cpt <= Vf) faire
    <Instruction(s)>;
    cpt ← cpt+1 ;
Fin-Tant-que;
```

Langage C :

```
cpt = Vi ;
while (cpt <= Vf)
{
    <Instruction(s)>;
    cpt = cpt+1 ;
}
```

Boucle répéter (Do...While)

La structure de contrôle répétitive **Répéter (do-while)** utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée on sort de la boucle, sinon on y accède.

Algorithme :

```
cpt ← Vi ;
Répéter
    <Instruction(s)>;
    cpt ← cpt+1 ;
Jusqu'à (cpt > Vf);
```

Langage C :

```
cpt = Vi ;
do
{
    <Instruction(s)>;
    cpt = cpt+1 ;
} while (cpt <= Vf);
```


Merci