# Filière IGE – 2017/2018
# Algorithmique & Programmation
# Leçon 4 : Traitements itératifs

Prof. ***A. DARGHAM***

## Traitements itératifs

- **Définition**
  - Un **traitement itératif** consiste à exécuter un ensemble d'instructions plusieurs fois.
  - En programmation, le traitement itératif se réalise en utilisant les **boucles** (***Loops*** en Anglais).
  - Le langage C offre trois types de boucles :
    - La boucle « **for** ».
    - La boucle « **while** ».
    - La boucle « **do … while** ».

# Traitements itératifs

- **Quelques exemples de traitements itératifs**
  - Pour créer une **table de multiplication**, de 4 par exemple, vous allez procéder comme si vous la récitiez :
    - $4 \times 1 = 4$
    - $4 \times 2 = 8$
    - …
    - $4 \times 9 = 36$
    - $4 \times 10 = 40$
  - Vous allez donc **multiplier 4 successivement par les nombres de 1 à 10**.
  - En algorithmique, vous connaissez les variables. Comment affecter une valeur de 1 à 10 successivement à une variable ? **C'est grâce à une boucle** !

# Traitements itératifs

- **Quelques exemples de traitements itératifs**
  - Et si maintenant vous vouliez créer l'ensemble des tables de multiplication : tables de 1, de 2, …etc, jusqu'à 10 ou plus ?
  - Il vous faudra **imbriquer deux boucles** !
  - Si vous voulez :
    - **calculer une puissance** quelconque;
    - **calculer une factorielle**;
    - **obtenir le plus grand des nombres** parmi une liste de nombres saisis (en attendant les tableaux);
    - …etc :
  - Il faudra encore utiliser les **boucles**.

## Boucle « for »

- **Syntaxes :**

```
for (expr1 ; expr2 ; expr3)
    Instruction;
```

Une seule instruction

```
for (expr1 ; expr2 ; expr3)
{
    Instruction_1;
    Instructio_2;
    . . . .
    Instruction_N;
}
```

Un bloc d'instructions

## Boucle « for »

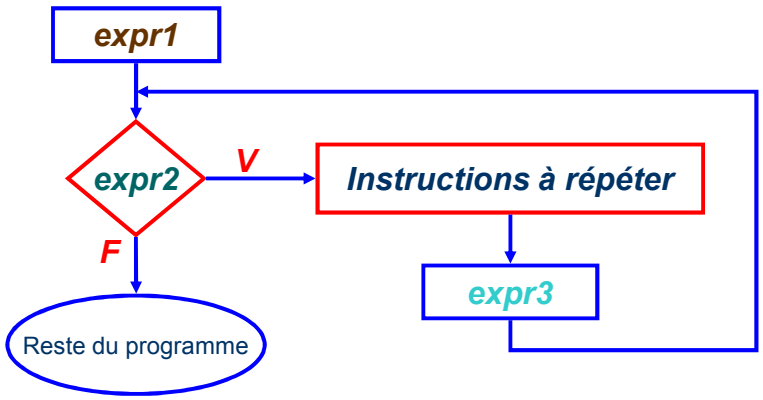
- **Rôles des expressions dans la boucle « for » :**
  - **expr1** : est une **expression d'initialisation** (d'un **compteur**). Généralement, c'est une opération d'affectation.
  - **expr2** : est une **expression booléenne**. Généralement c'est une **condition** qui détermine quand le **compteur** va s'arrêter.
  - **expr3** : est une **expression de progression** (**incrémentation** ou **décrémentation** du **compteur**).

## Boucle « for »

- **Exécution de la boucle « for » :**
  1. On exécute « **expr1** » en premier lieu et une seule fois.
  2. On teste « **expr2** » : si elle vraie, on exécute les instructions de la boucle for, puis on exécute « **expr3** ». Si elle est fausse on quitte la boucle for.
  3. On répète (2.) jusqu'à ce que « **expr2** » devienne fausse.

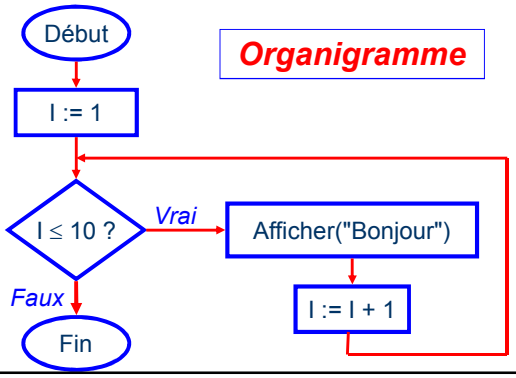
## Boucle « for »

- **Organigramme de la boucle « for » :**



# Boucle « for »

- **Exemple 1 :**
  - Afficher 10 fois le message "bonjour" :




*Organigramme*

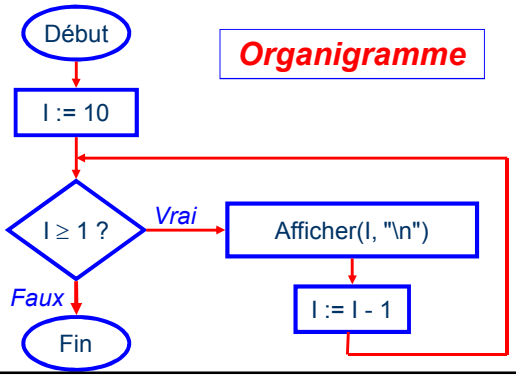
# Boucle « for »

```
#include <stdio.h>
main()
{
    int i;    /* compteur */
    for(i = 1; i <= 10; i++)
       printf("Bonjour\n");
    system("pause");
}
```

*Programme C*

# Boucle « for »

- **Exemple 2 :**
  - Afficher les entiers de 10 à 1 :




*Organigramme*

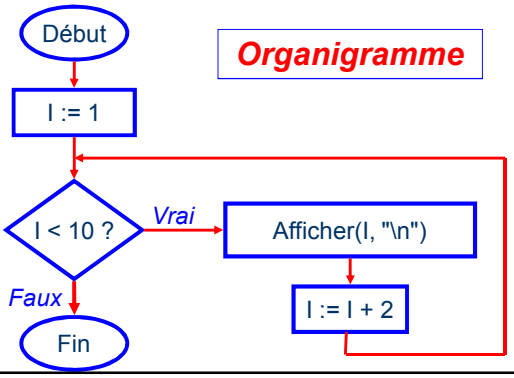
# Boucle « for »

```c
#include <stdio.h>
main()
{
    int i;    /* compteur */
    for(i = 10; i >= 1; i--)
       printf("%d\n", i);
    system("pause");
}
```

*Programme C*

# Boucle « for »

- **Exemple 3 :**
  - Afficher les entiers impairs de 1 à 10 :




*Organigramme*

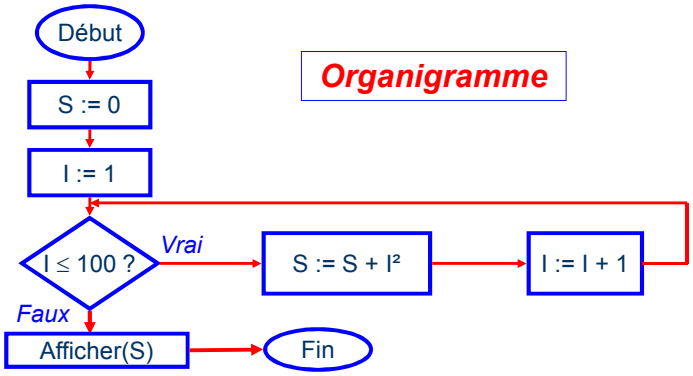
# Boucle « for »

```c
#include <stdio.h>
main()
{
    int i;    /* compteur */
    for(i = 1; i < 10; i += 2)
       printf("%d\n", i);
    system("pause");
}
```

*Programme C*

# Boucle « for »

- **Exemple 4 :**
  - Calculer la somme $1^2 + 2^2 + \ldots + 100^2$ :




*Organigramme*

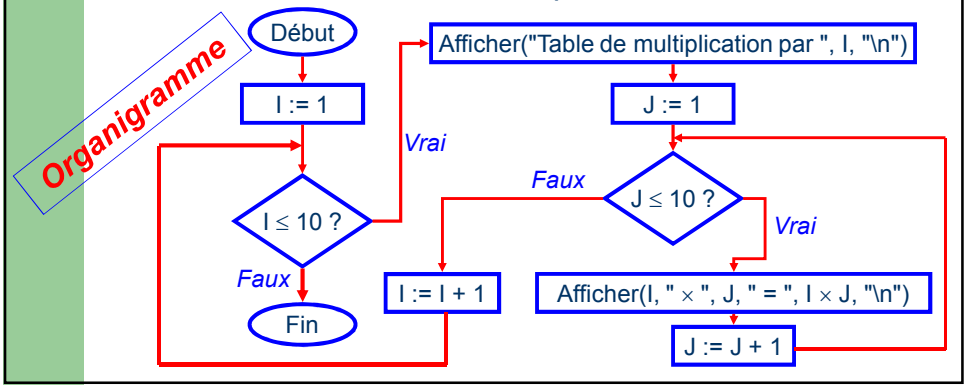
# Boucle « for »

```
#include <stdio.h>
main()
{
    int i, s = 0;
    for(i = 1; i <= 100; i++)
        s += i * i;
    printf("s = %d\n", s);
    system("pause");
}
```

*Programme C*

# Boucle « for »

- **Exemple 5 :**
  - Afficher les tables de multiplication de 1 à 10.




# Boucle « for »

**Programme C**

```c
#include <stdio.h>
main()
{
  int i, j;
  for(i = 1; i <= 10; i++)
  {
     printf("Table de multiplication par %d\n", i);
     for(j = 1; j <= 10; j++)
        printf("%d x %d = %d\n", i, j, i * j);
     printf("\n");
  }
  system("pause");
}
```

## Boucle « while »

- **Syntaxes :**

```
while (expr)
    Instruction;
while (expr)
{
    Instruction_1;
    Instructio_2;
    . . . .
    Instruction_N;
}
```

## Boucle « while »

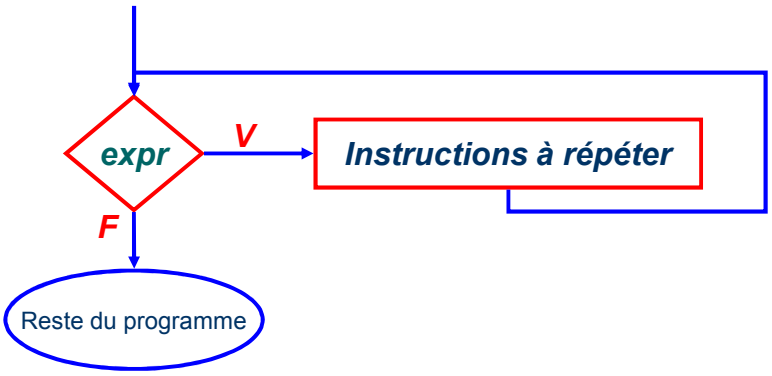
- La boucle **While** permet la répétition d'un bloc d'instructions **tant que la condition testée** définie par « **expr** » est **vérifiée**.
- Lors de l'exécution du programme, celui-ci arrive sur l'instruction **while**.
- Il évalue **l'expression booléenne** « **expr** ».
- Si l'expression retourne **VRAI** (valeur $\neq 0$), alors le programme exécute les instructions de la boucle.

## Boucle « while »

- Arrivé ici, il remonte au **while** et évalue de nouveau l'expression booléenne, si c'est VRAI alors il exécute de nouveau les instructions, et ainsi de suite, tant que l'expression retourne VRAI.
- Si l'expression devient fausse, alors le programme saute à l'instruction située juste après le **while**.

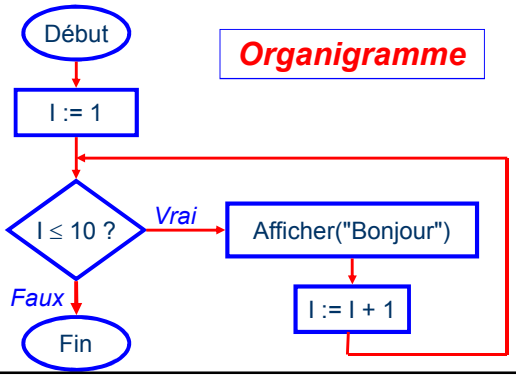
## Boucle « while »

- **Organigramme de la boucle « while » :**



# Boucle « while »

- **Exemple 1 :**
  - Afficher 10 fois le message "bonjour" :




*Organigramme*

# Boucle « while »

```c
#include <stdio.h>
main()
{
   int i;
   i = 1;
   while(i <= 10)
   {
      printf("Bonjour\n");
      i++;
   }
   system("pause");
}
```

*Programme C*

# Boucle « while »

- **Remarque** :
  - On peut transformer une boucle **for** en une boucle **while** en utilisant la transformation suivante :

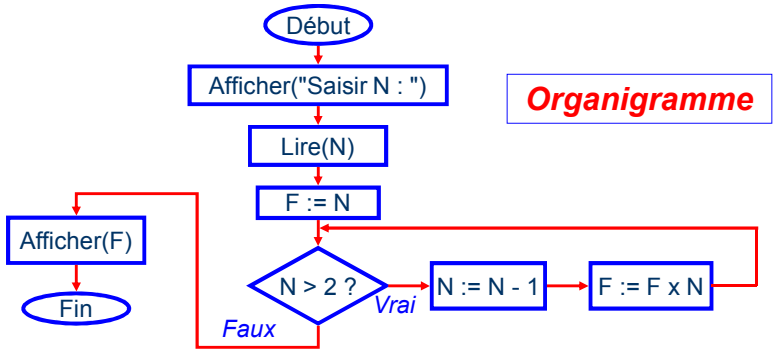
```
for(expr1 ; expr2 ; exp3)
    { /* bloc de for */ }
```

  - Sera transformé à :

```
expr1;
while(expr2)
    {
/* bloc de for */
    expr3;
    }
```

# Boucle « while »

- **Exemple 2 :**
  - Calculer la factorielle d'un entier N > 0 saisi :




*Organigramme*

# Boucle « while »

```c
#include <stdio.h>
main()
{
   int N, F;
   printf("Saisir N : ");
   scanf("%d", &N);
   F = N;
   while(N > 2)
   {
      N--;    F *= N;
   }
   printf("F = %d\n", F);
   system("pause");
}
```

***Programme C***

# Boucle « while »

- La boucle **while** est une boucle de ***pré-test***.
- *Cela signifie **qu'elle teste sa condition avant chaque itération** (répétition).*
- Dans l'exemple 1, la variable i est initialisée avec la valeur 1.
- Si i avait été initialisée à une valeur supérieure à 10, la boucle **while** ne serait jamais exécutée :

```c
i = 11;
while(i <= 10)
{
   /* elle ne sera jamais exécutée*/
   printf("Bonjour\n");
   i++;
}
```

## Boucle « while »

- Une caractéristique importante de la boucle **while** est qu'elle ne sera jamais exécutée si son **expression de test** (sa condition) est **initialement fausse**.
- Si vous voulez être sûre que la boucle **while** s'exécute pour la première fois, vous devez bien initialiser les données nécessaires.
- Règle (1) pour l'utilisation de la boucle **while** :

***Bien initialiser les données nécessaires qui assurent l'exécution de la boucle pour la première fois (s'il est nécessaire).***

## Boucle « while »

- Dans tous les cas (sauf pour certains cas rares), une boucle **while** doit **contenir une instruction qui force l'arrêt de son exécution**.
- Cela signifie qu'**on doit avoir une instruction à l'intérieur de la boucle qui garantie que la condition devienne fausse à un moment précis**.
- C'est la règle de terminaison de la boucle **while**.
- Si cette règle n'est pas respectée, on risque de tomber sur une **boucle infinie**, et l'**exécution du programme ne se termine jamais**.

## Boucle « while »

```
i = 1;
while(i <= 10)
{
   printf("Bonjour\n");
}
/* boucle infinie */
```

- C'est une boucle infinie, car elle ne contient pas une instruction qui modifie la valeur de i.
- En effet, à tout instant, la condition (i <= 10) est toujours vérifiée.

## Boucle « while »

- Il est aussi **possible de créer une boucle infinie** d'une **manière accidentelle** **en plaçant un** « ; » après la première ligne de la boucle **while**.
- Voici un exemple :

```
i = 1;
while(i <= 10);   /* Attention au ; */
{
    printf("Bonjour\n");
    i++;
}
```

- C'est **une boucle infinie**.

## Boucle « while »

- Encore, il **faut faire attention aux accolades**.
- Voici un exemple :

```
i = 1;
while(i <= 10)
    printf("Bonjour\n");
    i++;
```

- C'est **une boucle infinie**.

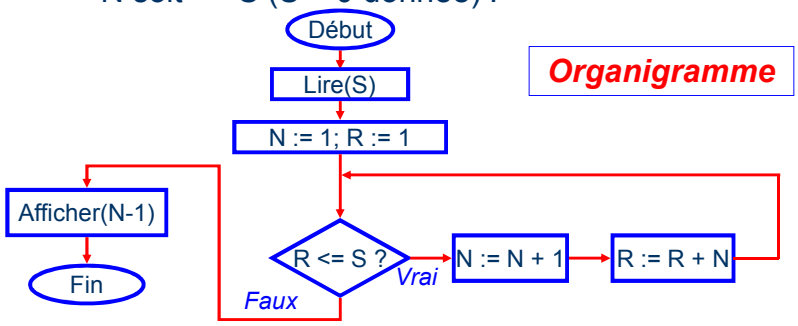
## Boucle « while »

- Une autre **erreur fatale avec les boucles**, est l'utilisation accidentelle de **l'opérateur d'affectation** « **=** » à la place de **l'opérateur d'égalité** « **==** » pour écrire une condition d'égalité.
- La boucle suivante est une **boucle infinie** :

```
while(r = 1) /* Erreur : Noter l'affectation */
{
   printf("Enter un nombre : ");
   scanf("%d", &n);
   r = n % 2;
}
```

# Boucle « while »

- **Exemple 3 :**
  - Trouver le plus petit nombre N tel que 1 + 2 + … + N soit <= S (S > 0 donnée) :




*Organigramme*

# Boucle « while »

*Programme C*

```c
#include <stdio.h>
main()
{
   int N, S, R;
   printf("Saisir S : ");
   scanf("%d", &S);
   N = R = 1;  /* affectations en cascade */
   while(R <= S)
   {
      N++;   R += N;
   }
   printf("%d\n", N - 1);
   system("pause");
}
```

# Boucle « while »

*Programme C*

```c
#include <stdio.h>
main()
{
   int N, S, R;
   printf("Saisir S : ");
   scanf("%d", &S);
   N = R = 1;  /* affectations en cascade */
   while(1)    /* Tant que c'est vraie ! */
   {
      N++;   R += N;
      if(R > S)
        break;
   }
   printf("%d\n", N - 1);
   system("pause");
}
```

# Boucle « while »

- L'instruction **break** à **l'intérieur d'une boucle** (for, while ou do-while) **permet l'arrêt immédiat** de la **boucle**, **sans même exécuter les instructions restantes dans le bloc de la boucle**.
- Après une instruction **break** dans une boucle, le **contrôle d'exécution** passe **immédiatement à la première instruction à l'extérieur de la boucle**.

## Boucle « while »

- Comme l'instruction **break**, l'instruction **continue** permet **également d'ignorer le reste des instructions de la boucle**, sans pour autant la quitter définitivement.
- En effet, après une telle instruction, **le contrôle d'exécution est automatiquement passé à l'itération suivante**.

## Boucle « while »

*Programme C*

```
#include <stdio.h>
main()
{
   int N;
   while(1)     /* Tant que c'est vraie ! */
   {
      printf("Saisir un entier : ");
      scanf("%d", &N);
      if(N % 2 == 0)  continue;
      if(N % 3 == 0)  break;
      printf("\tEn haut de la boucle.\n");
   }
   printf("\tA l'exterieur de la boucle.\n");
   system("pause");
}
```

## Boucle « while »

```
Saisir un entier : 7
     En haut de la boucle.
Saisir un entier : 4
Saisir un entier : 9
     A l'extérieur de la boucle.
```

## Boucle « do-while »

- **Syntaxes :**

```
do
    Instruction;
while(expr);
do
{
    Instruction_1;
    ....
    Instruction_N;
}while(expr);
```

## Boucle « do-while »

- La boucle **do-while** permet la répétition d'un bloc d'instructions **tant que la condition testée** définie par « **expr** » est **vérifiée**.
- Lors de l'exécution du programme, celui-ci arrive sur l'instruction **do-while**.
- Il exécute les instructions de la boucle **pour la première fois** (c'est ici la différence avec le **while**).
- Ensuite, il évalue **l'expression booléenne** « **expr** ».
- Si l'expression retourne **VRAI** (valeur $\neq 0$), alors le programme exécute les instructions de la boucle une deuxième fois, et ainsi de suite.

## Boucle « do-while »

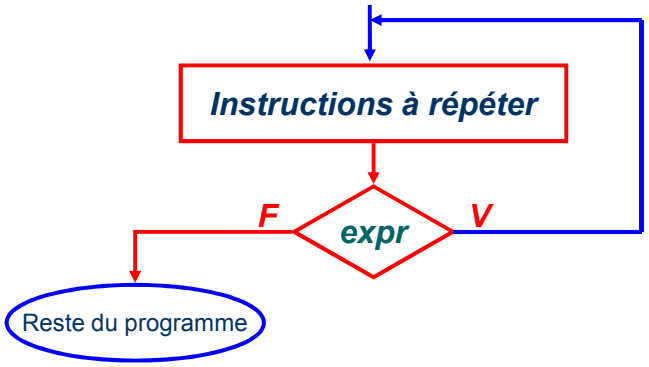
- Si l'expression devient fausse, alors le programme saute à l'instruction située juste après le **do-while**.
- La boucle **do-while** est une boucle de ***post-test***.
- *Cela signifie **qu'elle ne teste sa condition qu'après l'exécution complète de la première itération***.
- L'exemple suivant montre cette différence entre la boucle **while** et la boucle **do-while** :

## Boucle « do-while »

```
i = 11;
while(i <= 10)
{/* cette boucle ne sera pas exécutée */
   printf("Bonjour\n");
   i++;
}
i = 11;
do
{/* cette boucle sera exécutée au moins
    une fois */
   printf("Bonjour\n");
   i++;
}while(i <= 10);
```

## Boucle « do-while »

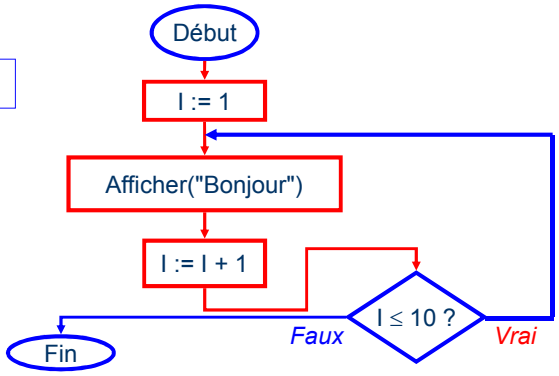
- Organigramme de la boucle « do-while » :



# Boucle « do-while »

- **Exemple 1 :**
  - Afficher 10 fois le message "bonjour" :

***Organigramme***




# Boucle « do-while »

***Programme C***

```
#include <stdio.h>
main()
{
   int i;
   i = 1;
   do
   {
      printf("Bonjour\n");
      i++;
   }while(i <= 10);
   system("pause");
}
```