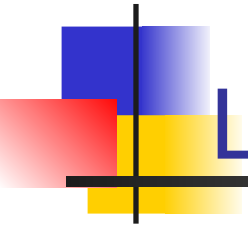


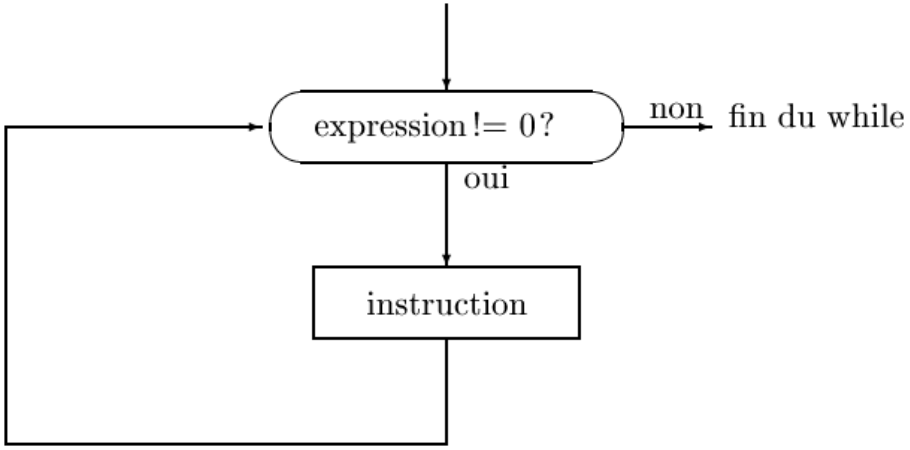
Les instructions de contrôle - boucles non déterministes

Rappel

- Lorsque l'on souhaite répéter plusieurs fois le même traitement (plutôt que de copier *n* fois un morceau de programme) :
  - On utilise les boucles
- Il existe deux types de boucles
  - Boucles déterministes
    - Le nombre d'itérations est connu ⇒ **`for`** `(...;...;...)`
  - Boucles non déterministes
    - Le nombre d'itérations dépend d'une condition[2]

L'instruction while

```
while (expression)
{
    bloc
d'instructions;
```



- Principe
  - Tant que expression est VRAI exécuter le bloc d'instructions entre accolades
- Fonctionnement
  - Evaluation de l'expression
    - Si VRAI
      Execution du while
    - Si FAUX
      Sortir de la boucle

Exemple

```
#include <stdio.h>
int main(void)
{
   int tab[3] = {23,56,78};
   int i,a;
   int somme = 0;
   i = 0;
   a = 3;

   while(i<a)
   {
      somme = somme + tab[i];
      i = i + 1;
   }
}
```

itération 1

itération 2

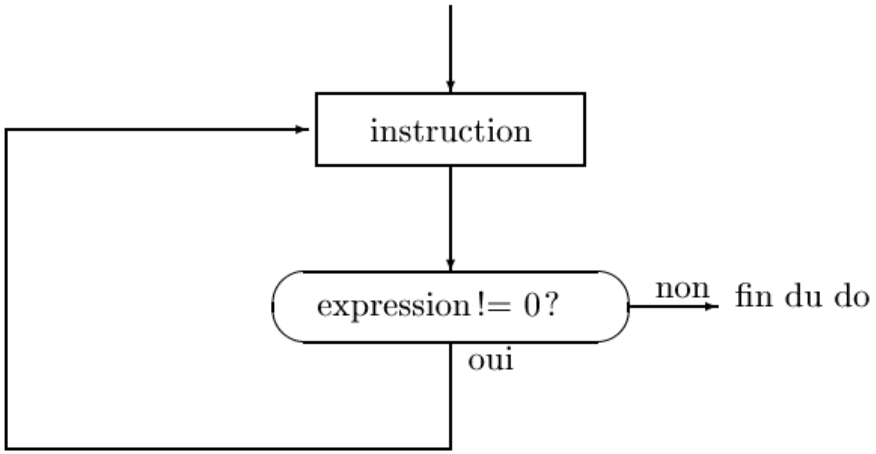
itération 3

- Fonctionnement pas à pas
  - *i possède la valeur* 0
  - *a possède la valeur* 3
  - *Evaluation de l'expression (i<a) ou 0<3*
  - ***Exécution du while*** vrai
    - *somme=* *pour i=0*
    - *incrémentation de la valeur de i* 23 *i=1*
  - *Evaluation de l'expression (i<a) ou 1<3*
  - ***Exécution du while***
    - *Somme=* *pour i=1*
    - *incrémentation de la valeur de i* 79 *i=2* vrai
  - *Evaluation de l'expression (i<a) ou 2<3*
  - ***Exécution du while***
    - *somme=* *pour i=2* vrai
    - *incrémentation de la valeur de i* 157 *i=3*
  - *Evaluation de l'expression (i<a) ou 3<3*
  - *FIN* FAUX

L'instruction do...while

Syntaxe:

```
do
{
   Bloc d'instructions
}
while (expression);
```



- Principe
  - Exécuter le bloc d'instructions tant que *expression* est **VRAIE**
- Fonctionnement
  - Exécution du bloc d'instructions **une première fois**
  - Evaluation de l'expression
    - Si **VRAI** alors nouvelle exécution du bloc d'instructions
    - Si **FAUX** , sortir de la boucle

Attention: Ne pas oublier le ;

Exemple

```
int main(void)
{
     int a, somme, compteur;
     compteur = 0;
     somme    = 0;
     a        = 3;
     do
     {
          somme    = somme + compteur;
          compteur = compteur + 1;
     }
     while (compteur<a);
}
```

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - Les variables a, somme, compteur sont **déclarées**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
	a=?
	somme=?
	compteur=?

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - la variable compteur est **initialisée**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=?	a=?
somme=?	somme=?
compteur=?	compteur=**0**

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - la variable somme est **initialisée**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=?	a=?
somme=?	somme= **0**
compteur=0	compteur=0

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

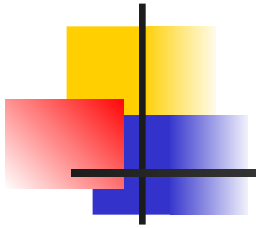
- Fonctionnement pas à pas
  - la variable a est **initialisée**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=?	a= **3**
somme=0	somme=0
compteur=0	compteur=0

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'instruction do-while est exécutée:
  - somme+compteur= **0**
  - somme prend la valeur de somme+compteur cad **0**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=0	somme= **0**
compteur=0	compteur=0

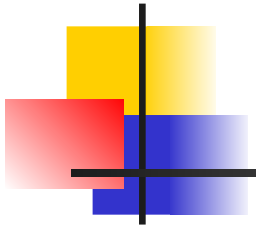


```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - la variable compteur est

    incrémentée

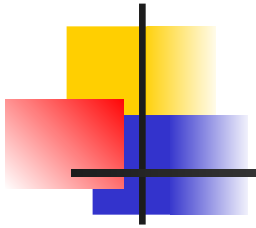
Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=0	somme=0
compteur=0	compteur=**1**



```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'expression compteur<a est évaluée
  - (1<3)? **VRAI**
  - **Reprendre la boucle**

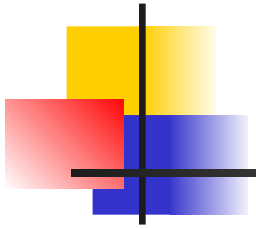
Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=0	somme=0
compteur=0	compteur=1



```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'instruction do-while est exécutée:
  - somme+compteur= **1**
  - somme prend la valeur de somme+compteur cad **1**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=0	somme=**1**
compteur=1	compteur=1



```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

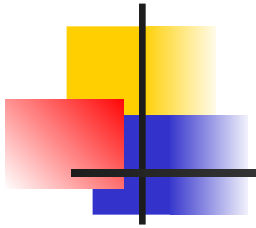
- Fonctionnement pas à pas
  - la variable compteur est **incrémentée**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=1	somme=1
compteur=1	compteur= **2**

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'expression compteur<a est **évaluée**
  - (2<3)? **VRAI**
  - **Reprendre la boucle**

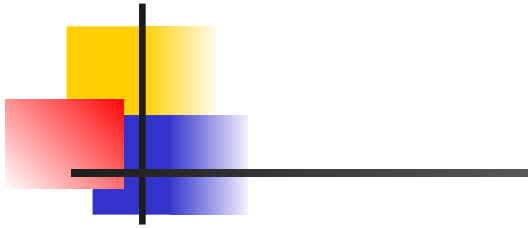
Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=1	somme=1
compteur=2	compteur=2



```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'instruction do-while est exécutée:
  - somme+compteur= **3**
  - somme prend la valeur de somme+compteur cad **3**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=1	somme= **3**
compteur=2	compteur=2



```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - la variable compteur est **incrémentée**

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=3	somme=3
compteur=2	compteur=**3**

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - L'expression compteur<a est évaluée
  - (3<3)? **FAUX**
  - **Arrêter la boucle**

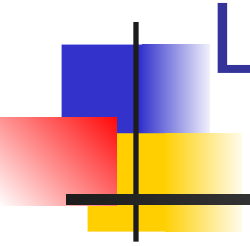
Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=3	somme=3
compteur=3	compteur=3

```
int main(void)
{
    int a, somme, compteur;
    compteur = 0;
    somme    = 0;
    a        = 3;
    do
    {
        somme    = somme + compteur;
        compteur = compteur + 1;
    }
    while (compteur<a);
}
```

- Fonctionnement pas à pas
  - Afficher la somme
  - Résultat à l'écran

    Somme = 3

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
a=3	a=3
somme=3	somme=3
compteur=3	compteur=3



Les instructions de contrôle -les instructions conditionnelles (suite)

L'instruction switch

```
switch (expression)
{
case constante1    :
    bloc d'instructions1 ;

case     constante2:
    bloc d'instructions2;
    ..
case constanteN      :
        bloc d'instructionsN;
        break;
default:
    bloc d'instructions
}
```

- Principe
  - structure à choix multiples
  - En fonction de la valeur de l'expression, exécuter le bloc correspondant jusqu'à l'instruction

Exemple 1

```
int main(void)
{
    int A=45,B=38,C;

    int chiffre=2;

    switch(chiffre)
    {
        case 0: C = A + B;
                break;

        case 2: C = A - B;
                break;

        case 4: C = A * B;
                break;

        case 6: C = A / B;
                break;

        default: C = A;
    }
}
```

Exemple 2

```
int main(void)
{
  char chaine[] = "greve a la SNCF le
   13/11/2007";
  int i=0, nb_chiffre=0,
   nb_non_chiffre=0;
  while (chaine[i] != '\0')
  {
    switch(chaine[i])
    {
      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '7':
      case '6':
      case '8':
      case '9':
                nb_chiffre++;
                break;
      default:  nb_non_chiffre++;
    }
    i++;
  }
```

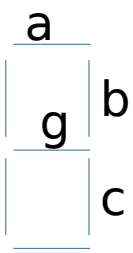
Exemple 3

```
int main(void)
{
  char chaine[] = "greve a la SNCF le 13/11/2007";
  int i=0, nb_chiffre=0, nb_non_chiffre=0;
  for(i=0; chaine[i] != '\0'; i++)
  {
    switch(chaine[i])
    {
      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '7':
      case '6':
      case '8':
      case '9':
                nb_chiffre++;
                break;
      default:  nb_non_chiffre++;
    }
  }
}
```

Exemple 4: décodeur 7 segments

Nous voulons écrire le décodeur 7 segments en C:

1 --> LED = 1111001
2 --> LED = 0100100
3 --> LED = 0110000
4 --> LED = 0011001
5 --> LED = 0010010
6 --> LED = 0000010
7 --> LED = 1111000
8 --> LED = 0000000
9 --> LED = 0010000



En utilisant SWITCH cet exemple s'écrit:

```
switch(nbr)
{
 case 1: LED = 0X79; break;
 case 2: LED = 0X24; break;
 case 3: LED = 0X30; break;
 case 4: LED = 0X19; break;
 case 5: LED = 0X12; break;
 case 6: LED = 0X02; break;
 case 7: LED = 0X78; break;
 case 8: LED = 0X00; break;
 case 9: LED = 0X10; break;
 default: LED = 0X40; //0
}
```

En utilisant IF ELSE cet exemple s'écrit:

```
if      (nbr == 1) LED = 0X79;
else if (nbr == 2) LED = 0X44;
else if (nbr == 3) LED = 0X30;
else if (nbr == 4) LED = 0X19;
else if (nbr == 5) LED = 0X12;
else if (nbr == 6) LED = 0X02;
else if (nbr == 7) LED = 0X78;
else if (nbr == 8) LED = 0X00;
else if (nbr == 9) LED = 0X10;
else
LED = 0X40;
```

- Et si nous voulons remplacer 'nbr' par 'nombre'!! -→Voilà l'avantage de « SWITCH »
- Et si on enlève les 'break'?!