

# Chapitre 5

## Structures de contrôle

# Plan

## A. Les instructions de branchements conditionnels

1. `if`

2. `if...else`

3. `switch`

## B. Les boucles

1. `while`

2. `do...while`

3. `for`

## C. Les instructions de branchements inconditionnels

1. `break, continue, goto, return, exit()`

## 5.1 Branchement conditionnel : **if**

```
if (<Condition>
    <Instruction>;
```

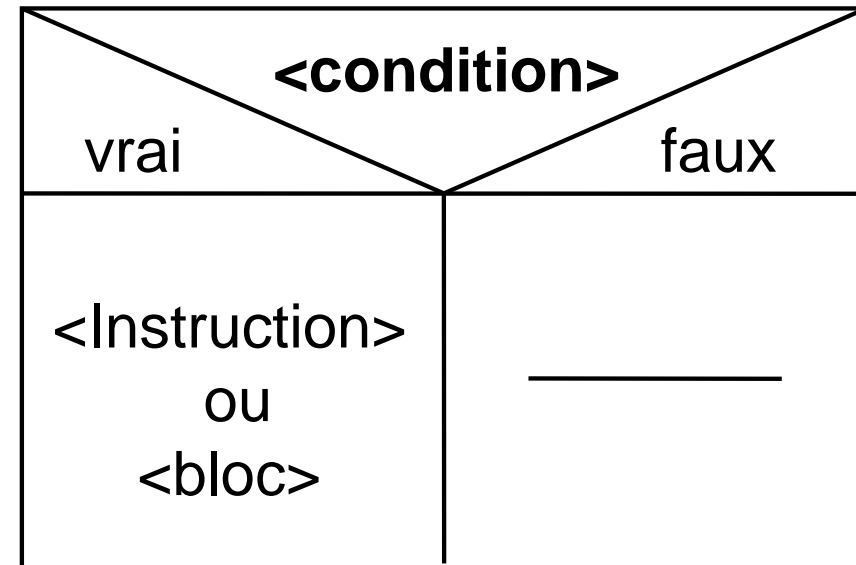
ou

```
if (<Condition>)
    <Bloc>
```

Exemple de <Bloc>

```
{
    <instruction>;
    <instruction>;
    ...
}
```

### Structogramme



**Attention à ne pas mettre de « ; »**

```
if (<condition>) ;
    printf(...);
```

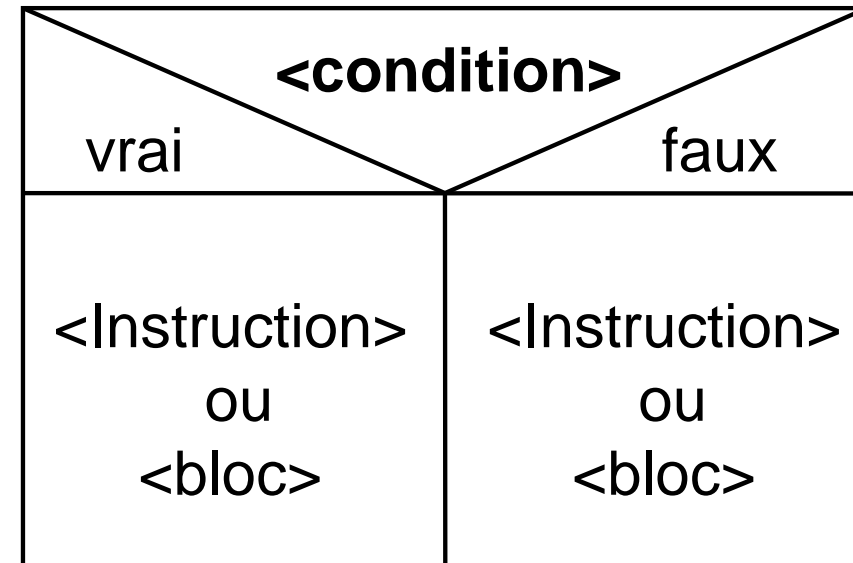
## 5.2 Branchement conditionnel : **if else**

```
if (<Condition>
    <Instruction>;
else
    <Instruction>;
```

ou

```
if (<Condition>)
    <Bloc>
else
    <Bloc>
```

### Structogramme



```
if (solde >= retrait)
    argent = retrait;
else
    argent = 0;
```

## 5.2 Exemple avec `if ... else`

### Calcul de l'évaluation ECTS en fonction de la moyenne

```
if (mean >= 5.3) {           // 5.3 <= mean
    evalECTS = 'A';
}
else {
    if (mean >= 4.8) {       // 4.8 <= mean < 5.3
        evalECTS = 'B';
    }
    else {
        if (mean >= 4.4) {   // 4.4 <= mean < 4.8
            evalECTS = 'C';
        }
        ...
    }
}
```



Toujours mettre les { }  
même s'il n'y a qu'une  
instruction

# Idem, avec une autre mise en page

```
if (mean >= 5.3)          // 5.3 <= mean <=6
    evalECTS = 'A';
else if (mean >= 4.8)     // 4.8 <= mean < 5.3
    evalECTS = 'B';
else if (mean >= 4.4)     // 4.4 <= mean < 4.8
    evalECTS = 'C';
...
else
    ...;
```



L'absence des { } rend le  
code moins lisible et  
moins maintenable

## 5.2 **if**, **else** : remarques

**La condition doit être entre parenthèses**

**if**(condition) et **else** ne se terminent pas par des ";"

{ et } définissent un bloc d'instructions

**else** n'existe pas sans **if**

**if** existe sans **else**

## 5.2 Styles de codage

*style Kernighan & Ritchie*

```
if (x == y) {  
    ..  
} else if (x > y) {  
    ...  
} else {  
    ....  
}
```

*style Whitesmiths*

```
if (x == y)  
{  
    ...  
}  
else if (x > y)  
{  
    ...  
}  
else  
{  
    ...  
}
```



# L'opérateur ternaire : **?:**

Aussi appelé *opérateur conditionnel*.

Il calcule une valeur selon le résultat d'une condition logique.

**<Condition> ? <Expression\_vraie> : <Expression\_fausse>**

```
int a = 3, b = 9, max, min;  
max = ((a > b) ? a : b);  
min = ((a > b) ? b : a);  
printf("Plus grande variable : %c", (a > b) ? 'a' : 'b');
```

```
int b = 28213;  
const char B_DIV_7 = (b%7==0)? 1:0;  
if(B_DIV_7) printf("'b' a une division entière par 7");
```

## 5.3 Branchement multiple : **switch**

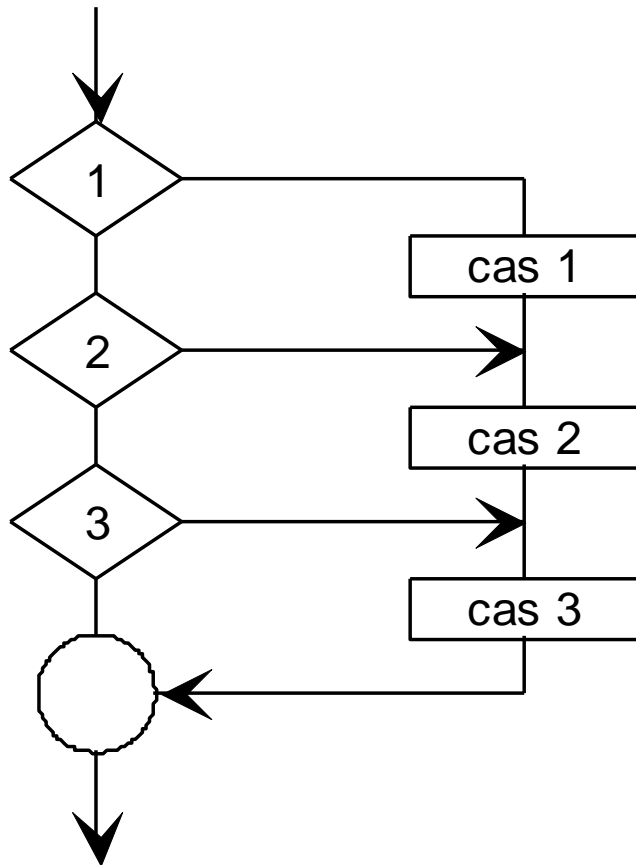
```
switch (value)
{
    case 1 : printf("Cas 1\n");
             value = 10;
    case 2 : printf("Cas 2\n");
             value = 20;
    case 3 : printf("Cas 3\n");
             value = 30;
}
```

Au début si <code>value</code> vaut	1	2	3	autre
Affichage	Cas 1 Cas 2 Cas 3	Cas 2 Cas 3	Cas 3	
À la fin <code>value</code> vaut	30	30	30	value

## 5.3 Syntaxe de **switch**

```
switch (<Paramètre>)  
{  
    case <valeur_A> : <Instruction_A1>;  
                     <Instruction_A2>;  
                     ...  
  
    case <valeur_B> : <Instruction_B1>;  
                     <Instruction_B2>;  
                     ...  
    [default:      <instruction(s)>]  
}
```

## 5.3 Organigramme du switch



**Programmation non structurée**  
(pas représentable par un structogramme)

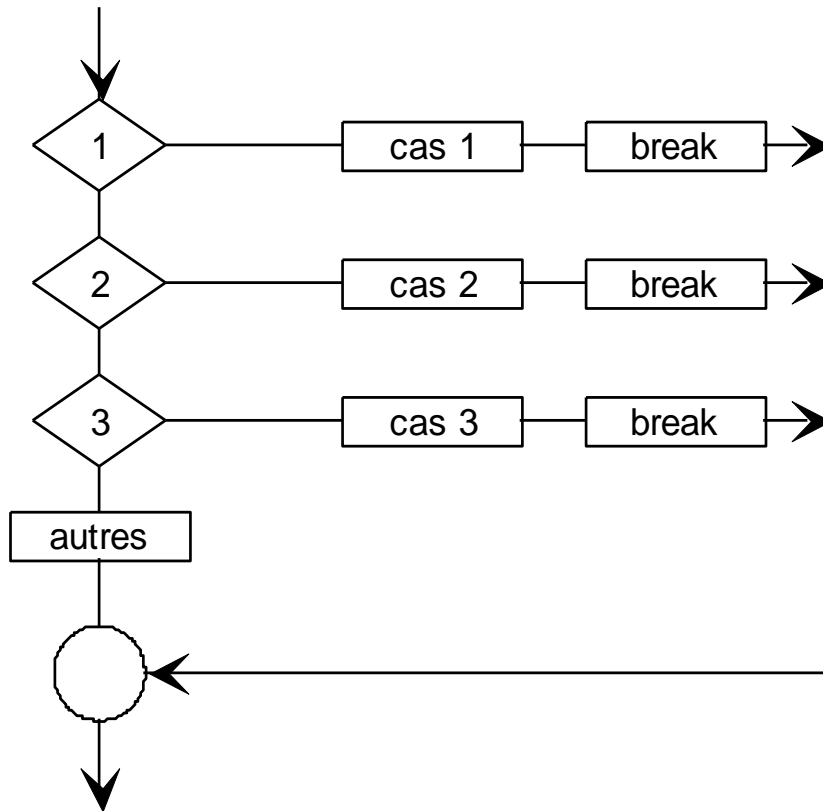
## 5.3 Exemple avec le mot-clé **break**

```
switch (value)
{
    case 1 : printf("Cas 1\n");
             value = 10;
             break;
    case 2 : printf("Cas 2\n");
             value = 20;
             break;
    case 3 : printf("Cas 3\n");
             value = 30;
             break;
    default: printf("Autres\n");
}
```

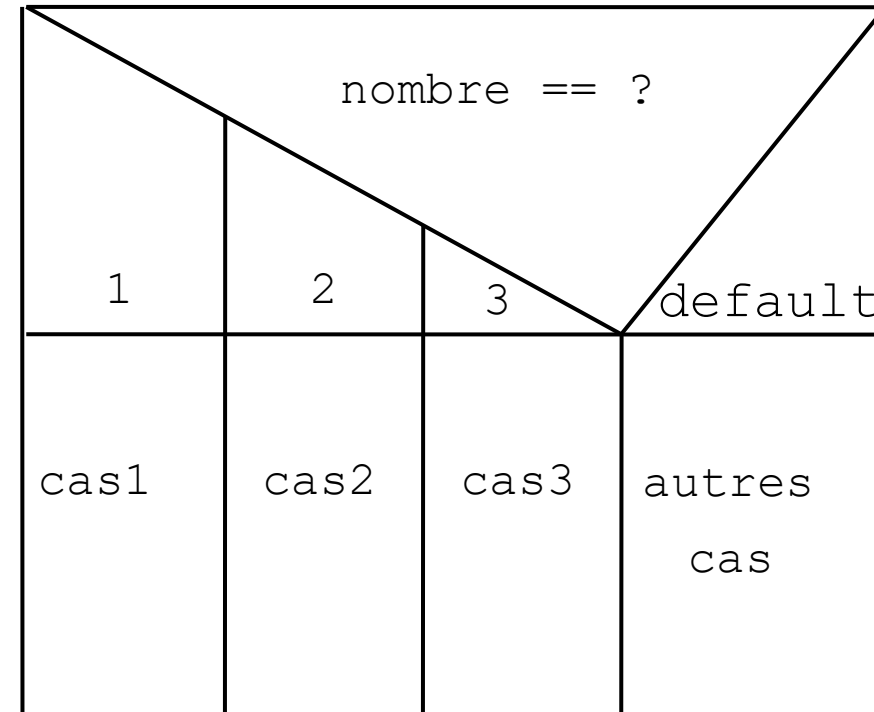
Au début si <code>value</code> vaut	1	2	3	autre
Affichage	Cas 1	Cas 2	Cas 3	Autres
À la fin <code>value</code> vaut	10	20	30	value

## 5.3 switch + break

Organigramme



Structogramme



## 5.3 Comparaison **switch** VS **if...else**

**switch** **permet de tester** des égalités entre valeurs **entières**

**switch** **ne peut pas tester** des valeurs **réelles**, *des chaînes de caractères*, des **structures**, des **tableaux**, des **intervalles de valeurs**

↪ Un **if-else** peut toujours remplacer un **switch**  
L'inverse n'est souvent pas possible

# Plan

## A. Les instructions de branchements conditionnels

1. `if`
2. `if...else`
3. `switch`

## B. Les boucles

1. `while`
2. `do...while`
3. `for`

## C. Les instructions de branchements inconditionnels

1. `break, continue, goto, return, exit()`



# Les boucles : motivations

Beaucoup de programmes **réalisent des actions répétitives**.  
Par exemple, pour créer un agenda, on donne chaque fois le nom, le prénom, le numéro de téléphone.

**NE PAS écrire un programme avec autant d'instructions qu'il y a de personnes.**

**Faire RÉPÉTER les instructions nécessaires à la saisie d'UNE personne**

## 5.4 Boucle **while**

**"Tant que la condition est vraie, répéter ..."**

Syntaxe

```
while (<Condition>)  
    <Instruction>;
```

```
while (<Condition >)  
{  
    <Bloc>  
}
```

Structogramme

Tant que <condition> vraie

```
<Instruction>;  
ou  
{ <Bloc> }
```

## 5.4 Exemple de boucle **while**

Calcul du modulo :  $n \% m$

**Solution** : tant que  $n \geq m$  , soustraire  $m$  à  $n$ .

```
int main(void)
{
    int n = 10, m = 3;
    while (n >= m)
    {
        n = n - m;
    }
    printf("modulo = %d", n)

    return 0;
}
```

## 5.4 Exemples de boucle **while**

quel sera l'affichage ?

**A**

```
i=0;
while(i++ < 3)
    printf(" %d ",i);
printf(" et %d ",i);
```

**B**

```
i=0;
while(++i < 3)
    printf(" %d ",i);
printf(" et %d ",i);
```

**C**

```
i=0;
while(i < 3)
    printf(" %d ",i++);
printf(" et %d ",i);
```

**D**

```
i=0;
while(i < 3)
    printf(" %d ",++i);
printf(" et %d ",i);
```



## Remarques

Avec **while**, tant que la condition est vraie, l'instruction ou le bloc suivant la condition (le *corps de la boucle*) est exécuté.

Le corps de la boucle **while** n'est donc pas forcément exécuté.

Les instructions dans le corps de la boucle doivent modifier la condition afin de la rendre fausse à un moment donné. C'est la condition de sortie de la boucle.

Sinon, on exécute une **boucle sans fin**.

## 5.5 Boucle **do...while**

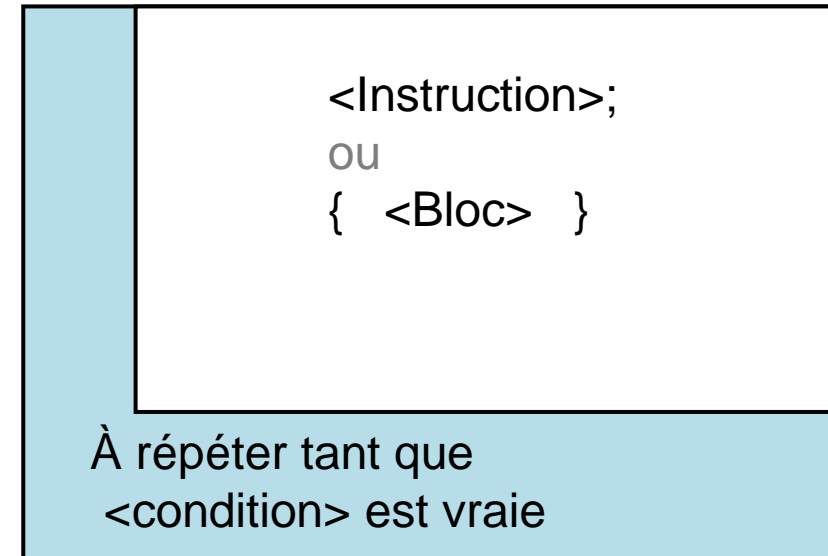
**"Répéter ... tant que la condition est vraie"**

### Syntaxe

```
do  
    <Instruction>;  
while (<Condition>);
```

```
do  
{  
    <Bloc>  
}  
while (<Condition>);
```

### Structogramme



## 5.5 Exemple de boucle **do...while**

Lire un nombre au clavier compris entre 0 et 10

```
int main(void)
{
    int nb;
    do
    {
        printf("Entrez un nombre (0-10) :");
        scanf(" %d", &nb);
    } while (nb < 0 || nb > 10);

    return 0;
}
```

# Remarques

La ou les instructions du corps de la boucle sont exécutées tant que la condition est vraie.

Le corps de la boucle **do...while** est exécuté **au moins une fois**.

Les instructions dans le corps de la boucle doivent modifier la condition pour la rendre fausse à un moment donné

```
do  
{
```

```
...
```

```
}while (condition)
```



**Doit se terminer par ";"**



## 5.6 Boucle **for**

"Répéter un certain nombre de fois ..."

**Syntaxe**

```
for (<Init>; <Condition>; <Incrémentation>)  
    <Instruction>;
```

```
for (<Init>; <Condition>; <Incrémentation >)  
{  
    <Bloc>  
}
```

**Structogramme**

```
for (<Initialisation>; <Condition>; <Incrémentation>)  
    <Instruction>;  
    ou  
    { <Bloc> }
```

# Remarques

Permet de répéter une ou plusieurs instructions, un **nombre de fois déterminé**.

Utilise un **indice** (un entier) pour compter le nombre de fois que la boucle est effectuée.

Sa **valeur** doit être

- **initialisée** au départ,
- **testée à chaque passage** pour savoir si le programme a effectué le nombre de boucles souhaitées,
- et **modifiée** (augmentée/diminuée) à chaque passage.

## 5.6 Boucle **for**

```
for (initialisation; condition; incrément)  
{  
    liste d'instructions;  
}
```

### **Initialisation-s**

initialise la variable *d'indice* une seule fois, à l'entrée dans la boucle

### **Condition-s**

doit être vraie pour que les instructions de la boucle s'exécutent

### **Incrément-s**

modifie la valeur de *l'indice*

## 5.6 Exemple

Calculer  $a^n = a * a * \dots * a$  pour  $n \geq 0$  en partant de  $a^0$  qui vaut 1

```
int main(void)
{
    int i, n = 3;
    double a = 2.0;
    double power = 1.0;

    for( i=0; i<n; i++ )
    {
        power = power * a;
    }

    return 0;
}
```

## 5.6 Fonctionnement de la boucle **for**

```
1          2, 5, 8  4, 7  
for (n = 0; n < 2; ++n) {  
    3 6  
    printf("Iteration:%d\n", n);  
}  
9 printf("Fin:%d", n);
```

**Itération:0**

**Itération:1**

**Fin:2**

n=2

# Commentaires

- (1) l'expression d'<Initialisation> est exécutée (1).
  - (2) <Condition> est testée (2).
  - (3) Si elle est remplie, alors, l'<Instruction> est exécutée (3),
  - (4) puis l'<Incrémentation> est effectuée (4)
  - (5) <Condition> est à nouveau testée (5).
  - (8) Si la <Condition> n'est pas remplie (8), la boucle for prend fin.
- L'instruction d'<Incrémentation> est effectuée une dernière fois avant de terminer la boucle (7).

La variable utilisée prend donc la prochaine valeur supérieure à la dernière autorisée par la <Condition>, sauf dans le cas où la <Condition> n'est d'emblée pas remplie.

# Valeur de l'indice



	init	fin	pas	Nb itération	Affichage
<b>for (i=0 ; i&lt;5 ; i++)</b> printf ("%d, ", i) ;	0	5	1	5	0,1,2,3,4,
<b>for (j=4 ; j&lt;=12 ; j=j+2)</b> printf ("%d, ", j) ;					
<b>for (c='a' ; c&lt;'f' ; c+=1)</b> printf ("%c, ", c) ;					
<b>for (k=5 ; k&gt;0 ; k--)</b> printf ("%d, ", k) ;					

## 5.6 Boucle **for** et boucle **while**

```
for ( i=1; i<132 ; i=i+2 )  
{  
    Instructions;  
}
```

...est équivalent à...

```
i=1;  
while ( i<132 )  
{  
    Instructions;  
    i=i+2;  
}
```

initialisation

condition

incrémentation



## 5.6 Exemple d'une boucle **for**

```
for (int i=0, j=0; i < 10; i++, j=i*i )  
{  
    printf("x = %2d, x^2 = %2d\n", i,j);  
}
```

```
x = 0, x^2 = 0  
x = 1, x^2 = 1  
x = 2, x^2 = 4  
x = 3, x^2 = 9  
x = 4, x^2 = 16  
x = 5, x^2 = 25  
x = 6, x^2 = 36  
x = 7, x^2 = 49  
x = 8, x^2 = 64  
x = 9, x^2 = 81
```

## 5.6 Exemples de boucles 'infinies'

```
for ( ; ; )  
{  
}
```

```
while (1)  
{  
}
```

## 5.6 Choisir LA bonne boucle

La boucle **for** est à privilégier quand on **connait à l'avance** le nombre d'itérations.

Si on **ne connait pas à l'avance** le nombre d'itérations à effectuer :

On choisit **do...while** quand le corps de la boucle doit être effectué **au moins une fois**.

On choisit **while** lorsqu'il ne faut pas obligatoirement entrer dans la boucle.

# Plan

## A. Les instructions de branchements conditionnels

1. `if`
2. `if...else`
3. `switch`

## B. Les boucles

1. `while`
2. `do...while`
3. `for`

## C. Les instructions de branchements inconditionnels

1. `break`, `continue`, `goto`, `return`, `exit()`

## 5.7 L'instruction **return**

L'instruction **return** termine l'exécution de la fonction courante.

Dans les programmes simples faits jusqu'ici, le code s'exécute dans la fonction **main()** et l'instruction **return** termine l'exécution du programme.

```
int Inc(int x)
{
    return ++x;
    printf("%d", x);
}
int main(void)
{
    int y = Inc(10);
    return 0;
}
```

## 5.7 La fonction **exit** (<int>)

La fonction **exit** permet de terminer l'exécution du programme où que l'on se trouve.

Elle agit de manière similaire à une instruction **return** depuis le programme principal **main**.

Elle prend un paramètre (code d'erreur entier) destiné à l'environnement d'exécution ( **0:ok** , -1:erreur, ...)

une directive **#include <stdlib.h>** est nécessaire

## 5.7 La fonction **exit** (<int>)

### Exemples

Le bon déroulement:

```
exit(0);  
exit(EXIT_SUCCESS); // 0
```

Une erreur lors du déroulement

```
exit(1);  
exit(EXIT_FAILURE); // 1
```

Définition de ses propres codes:

```
#define EX_OK      0           // success  
#define EX__BASE  64          // error messages  
#define EX_USAGE  65          // command usage error
```

## 5.7 La fonction **exit** (<int>)

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE * pFile;
    pFile = fopen("myfile.txt", "r");
    if (NULL == pFile)
    {
        printf ("Error opening file");
        exit(1);
    }
    return 0;
}
```



## 5.7 La fonction **exit** (<int>)

Utilisation de l'information envoyée par **exit()**

Dans un fichier de commande sous Windows (.bat)

```
test.exe  
IF ERRORLEVEL 1 ECHO erreur lecture
```

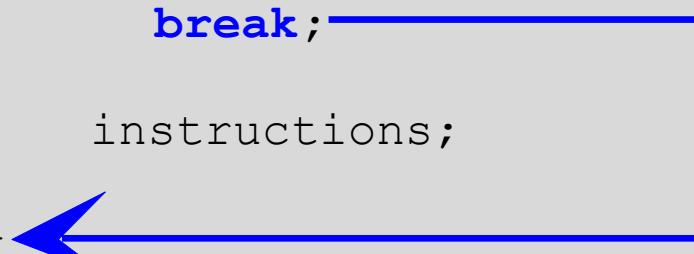
Sous Linux dans un fichier script bash

```
#!/bin/bash  
test  
if [ "$?" = "1" ]; then  
    echo " erreur lecture "  
fi
```

## 5.7 L'instruction **break**

### ⚠ **Programmation non structurée**

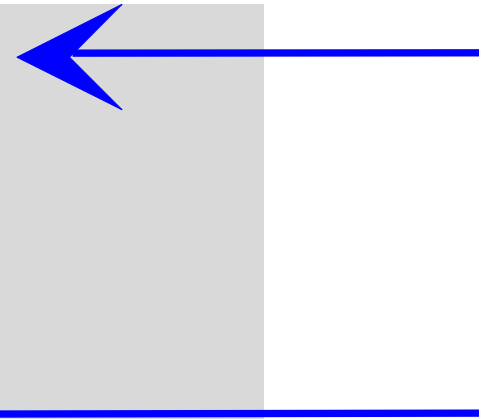
```
for (i = 0; i <= 20; i++)  
{  
  
    instructions;  
  
    if(condition)  
  
        break;  
  
    instructions;  
  
}
```



## 5.7 L'instruction **continue**

### ⚠ **Programmation non structurée**

```
while (condition w)
{
    instructions;
    if(condition i)
        continue;
    instructions;
}
```

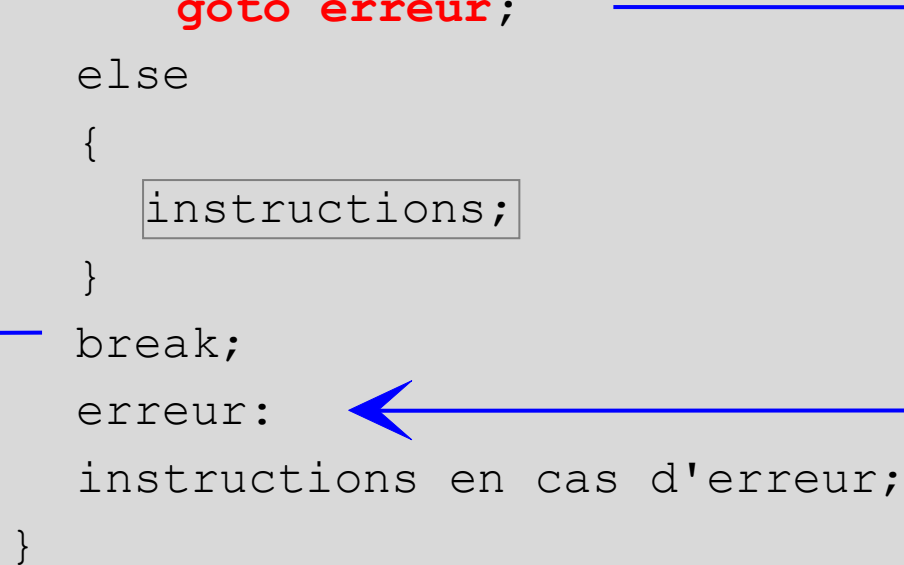


Passe au test w sans effectuer les instructions qui suivent

## 5.7 L'instruction `goto`

### ⚠ Programmation non structurée

```
{  
    instructions;  
    if(a > b)  
        goto erreur;  
    else  
    {  
        instructions;  
    }  
    break;  
    erreur:  
    instructions en cas d'erreur;  
}
```



# Mots réservés du langage C

En bleu, les mots-clés déjà vus → **plus de 50% !**

<code>_Alignas</code> <sup>(C11)</sup>	<code>_Alignof</code> <sup>(C11)</sup>	<code>_Atomic</code> <sup>(C11)</sup>	<code>_Bool</code> <sup>(C99)</sup>
<code>_Complex</code> <sup>(C99)</sup>	<code>_Generic</code> <sup>(C11)</sup>	<code>_Imaginary</code> <sup>(C99)</sup>	<code>_Noreturn</code> <sup>(C11)</sup>
<code>_Static_assert</code> <sup>(C11)</sup>	<code>_Thread_local</code> <sup>(C11)</sup>	<code>auto</code>	<code>break</code>
<code>case</code>	<code>char</code>	<code>const</code>	<code>continue</code>
<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>
<code>enum</code>	<code>extern</code>	<code>float</code>	<code>for</code>
<code>goto</code>	<code>if</code>	<code>inline</code> <sup>(C99)</sup>	<code>int</code>
<code>long</code>	<code>register</code>	<code>restrict</code> <sup>(C99)</sup>	<code>return</code>
<code>short</code>	<code>signed</code>	<code>sizeof</code>	<code>static</code>
<code>struct</code>	<code>switch</code>	<code>typedef</code>	<code>union</code>
<code>unsigned</code>	<code>void</code>	<code>volatile</code>	<code>while</code>

Référence: <http://en.cppreference.com/w/c/keyword>

Quel est leur rôle?

# Exercices



## Exercices du chapitre 05