# Chapitre 4 : les boucles

## 1. Introduction

Une boucle en anglais loop est une structure de contrôle, visant à exécuter un ensemble d'instructions plusieurs fois consécutives, elle est exécutée selon le cas, soit par un nombre connu de fois à l'avance (boucle itérative), soit jusqu'à ce qu'une condition permette la sortie de la boucle (boucle conditionnelle).
Il existe trois types de boucles :

- Une boucle conditionnelle avec une pré-condition : la condition est vérifiée avant la première itération.
- Une boucle conditionnelle avec une post-condition : la condition est vérifiée après la première itération.
- Boucle itérative : Un compteur permet de compter le nombre d'itérations.

Une erreur de programmation peut avoir pour conséquence que la condition de sortie ne soit jamais satisfaite. Cela fait que le programme s'exécute infiniment, on l'appelle la boucle infinie.

## 2. La boucle « Tant Que »

La boucle « Tant Que » est une boucle pré-conditionnelle dans laquelle un ensemble d'instructions est exécuté de manière répétitive sur la base d'une condition booléenne. La boucle « while » peut être vue comme une répétition de l'instruction « if ». elle est utilisée lorsque nous avons un ensemble d'instructions qui se répètent avec la possibilité qu'elles ne soient pas exécutées du tout (0 fois ou plus), selon la condition prédéfinie. La boucle se compose de deux parties :

- Condition: C'est une expression d'un type logique, dont la valeur est soit true vrai, soit false faux
- Bloc d'instruction : il est exécuté tant que la condition est vraie.

### 2.1.syntaxe

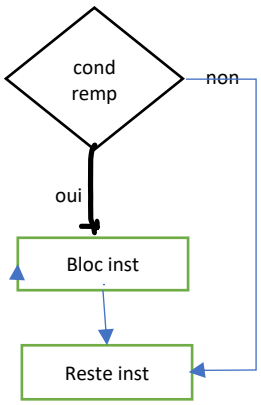
| Algorithme | C |
|---|---|
| **TantQue** Condition **Faire**<br>  Bloc d'instruction<br>**FinTantQue**<br>Le reste du programme | **while (**Condition**)**<br>**{**<br>  Bloc d'instruction<br>**}**<br>Le reste du programme |

**TantQue** ou **TQ**, **Faire** et **FinTantQue** ou **fTQ** sont des mots réservés dans l'algorithme. De même, pour **while** en C. la condition est toujours dans l'algorithme entre les mots TantQue et faire, tandis qu'en C elle est toujours entre parenthèses. Pour construire la condition, nous utilisons des comparaisons (>, <, =, ≠, ...) et des opérations logiques (et &&, ou || , non!, ...).
Les instructions de while en C sont met entre deux accolades {}, et peuvent être supprimées si elles ne contiennent qu'une seule instruction ({} facultatif). Si nous trouvons un ensemble d'instructions alors que nous ne trouvons pas les accolades, cela signifie que seule la première instruction est répétée.

**Observation**

- En C, le type booleen est exprimé par un int. Où le faux est exprimé par 0, et le vrai est exprimé par n'importe quel nombre différent de 0.
- Pas de « ; » après }.

## 2.2. Algorigramme




## 2.3. Exécution

Le processus d'exécution de la boucle TantQue commence par le calcule de l'expression de la condition, dont le résultat est de type booléen, donc si le résultat est vrai, le bloc d'instructions entre le mot Faire et FinTantQue dans l'algorithme, ou entre {} dans C est exécuté, puis nous ré-évaluons de nouveau le test et nous recommençons à zéro. Lorsque le résultat du test est faux, nous quittons la boucle en sautant au premiere instruction qui suit immédiatement la boucle. La condition est généralement appelée «condition de continuation».

**Observation**

Puisque la boucle While vérifie la condition avant la première itération, il est possible que la condition ne soit pas vérifiée la première fois, et donc ses instructions ne soient pas exécutées du tout.

## 2.4. Exemple

Écrivez un programme qui lit deux nombres entiers puis affiche le quotient du premier divisé par le second, sans utiliser l'opérateur de division (div, /).

**Remarque** La division est une soustraction successive.

| Algorithme | C | ecran |
|---|---|---|
| **algorithme** quotient<br>**var** x, y, q, r :entier<br>/*x premier nbr, y deuxième, q quotient, r reste*/<br>**début**<br>écrire("entrer 2 nbrs")<br>lire(x, y)<br>q←0<br>r←x<br>**TQ** r>y **Faire**<br>r←r-y<br>q←q+1<br>**FinTQ**<br>écrire("le quotient de", x, "sur", y, "est", q, "le reste est", r)<br>**fin** | #include <stdio.h><br>int main()<br>{<br>int x, y, q, r ;<br>printf("entrer 2 nbrs\n") ;<br>scanf("%d%d", &x, &y) ;<br>q=0 ;<br>r=x ;<br>**while** ( r>y )<br>**{**<br>r-=y ;<br>q++ ;<br>**}**<br>printf("le quotient de %d sur %d est %d le reste est %d\n", x, y, q, r) ;<br>} | entrer 2 nbrs<br>17 5<br>le quotient de 17 sur 5 est 3 le reste est 2 |

L'algorithme prend deux nombres x et y, et renvoie le quotient q et le reste r. Au début, supposons que le reste est x, et à chaque itération nous en diminuons le dénominateur « y » jusqu'à ce qu'il devienne inférieur au dénominateur, a chaque fois nous diminuons « y », nous ajoutons 1 au quotient q. La boucle TQ ne peut jamais être exécutée si x est inférieur à y dès le début. Dans ce cas q=0 et r=x.

**Exemple 2**

Écrivez un programme qui ne s'arrête pas tant que l'utilisateur n'appuie pas sur la touche Entrée.

```
#include <string.h>
int main()
{
```

```
 while (getchar() != '\n');
 return 0;
}
```

# 3. La boucle Faire…Tant Que

Avec la boucle Tant Que, il est possible que la boucle ne soit pas exécutée du tout, si la condition est fausse dès la première évaluation. Mais dans certains cas, le programmeur veut s'assurer que la boucle est exécutée au moins une fois. La boucle « **Faire...TantQue** » est une boucle post-conditionnelle dans laquelle un ensemble d'instructions est exécuté de manière répétitive, sur la base d'une condition booléenne. elle est utilisée lorsque nous avons un ensemble d'instructions répétées et exécutées au moins une fois, quelle que soit la condition (1 ou plus). La boucle se compose de deux parties :

- Bloc d'instruction : Il est exécuté tant que la condition est vraie, sauf la première fois qu'il est exécuté quelle que soit la condition.
- Condition : une expression de type booléen dont la valeur est soit vrai soit faux.

## 3.1.syntaxe :

| Algorithme | C |
|---|---|
| **Faire**<br>  Bloc d'instruction<br>**TantQue** Condition<br>Le reste des instructions | **do {**<br>  Bloc d'instruction<br>**}**<br>**while (**Condition**) ;**<br>Le reste des instructions |

Les mots **Faire** et **TantQue** sont des mots réservés dans l'algorithme. **do** et **while** en **C**. La condition est toujours en dernier après TantQue, et en **C** elle est toujours entre parenthèses (). Pour construire la condition on utilise des opérations de comparaison (>, <, =, ≠, ...) et des opérations logiques (et &&, ou ||, non !, ...).
Les instructions de « **do…while »** en **C** sont mis à l'intérieur de deux accolades {} à l'intérieur de **do** et **while**. {} peut être omis, s'il ne contient qu'une seule instruction ({} facultatif).

**Observation :**

- La boucle « **do…while** » se termine toujours par un point-virgule « ; »

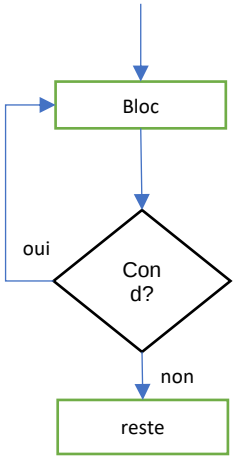
On peut exprimer le « **faire…TantQue** » comme **Répéter…Jusqu'à** (jusqu'à ce que la condition soit satisfaite). Dans ce cas, la condition devient une condition d'arrêt et non une condition de continuité, qui est la négation de la condition de la boucle de **TantQue**.

**Exemple :**

| faire…TantQue | Répéter…Jusqu'à |
|---|---|
| **Faire**<br>  Bloc d'instruction<br>**TantQue** x>y<br>Le reste des instructions | **Répéter**<br>  Bloc d'instruction<br>**Jusqu'à** x≤y<br>Le reste des instructions |

**Notez que** la négation de > est ≤.

## 3.2. Algorigramme



### 3.3. Exécution

Le processus d'exécution de la boucle conditionnelle « **Faire...TantQue** » exécute le bloc d'instructions qui se trouve entre le mot **Faire** et **TantQue** dans l'algorithme, ou entre **do** et **while** en **C**, puis on calcule l'expression de condition, dont le résultat est d'un type booléen, si le résultat est vrai, il exécute à nouveau le bloc des instructions, et ainsi de suite jusqu'à ce que le résultat du test soit faux, on quitte la boucle a l'instruction qui le suit immédiatement.

**Observation :**

Puisque la boucle « **Faire...TantQue** » exécute les instructions de la première itération avant que la condition ne soit vérifiée, la boucle exécute au moins une itération, même si la condition n'est pas satisfaite dès le départ.

### 3.4. Exemple :

Écrivez un programme qui lit un ensemble d'entiers à l'aide d'une seule variable, s'arrête au premier 0 qui le lit, puis affiche le nombre des entiers saisis.

**Algorithme**

```
algorithme lireNbrs
 var x, nb :entier
/*x pour lire nbrs, nb pour compter les nbrs*/
début
  nb←0
  Faire
    écrire("entrer un nbr")
    lire(x)
    nb←nb+1
  TQ x≠0
  écrire("Le nombre de nombres est", nb-1)
fin
```

**C**

```
#include <stdio.h>
int main()
{
 int x, nb ;
 nb=0 ;
 do {
   printf("entrer un nbr") ;
   scanf("%d", &x) ;
   nb++ ;
 }
 while ( x!=0 ) ;
 printf("Le nombre de nombres est %d", nb-1) ;
}
```

**Ecran**

```
entrer un nbr: 5
entrer un nbr: 7
entrer un nbr: -2
entrer un nbr: 0
Le nombre de nombres est 3
```

L'algorithme a besoin de la variable x pour lire les nombres et de la variable nb pour compter les nombres. On met nb 0 comme valeur initiale, puis on saisie un nombre x et on ajoute 1 a nb, si x est 0, on arrête, sinon on répète la boucle jusqu'à ce que l'utilisateur saisie le nombre 0. La boucle s'exécutera à moins une fois. Enfin, on montre la valeur de nb-1 pour que le nombre 0 ne soit pas compté.

## 4. La boucle « pour »

La boucle « **pour** » (**for**) est une boucle itérative inconditionnelle dans laquelle un ensemble d'instructions est exécuté de manière itérative un nombre de fois prédéterminé. La boucle se compose de deux parties :

- Compteur : Pour compter le nombre d'itérations, c'est une variable de type entier ou caractère. Où il définit sa valeur initiale, sa valeur finale et la méthode pour l'incrementer ou la décrémenter.
- Bloc d'instructions : à exécuter dans chaque itération.

### 4.1.syntaxe de « pour » en algorithme

**Algorithme**

```
Pour compteur←v_init à v_final pas pas Faire
  Bloc d'instructions
FinPour
Le reste des instructions
```

Les mots **Pour**, **jusqu'à** ou **à**, **faire** et **FinPour** ou **FPour** sont des mots réservés dans l'algorithme.

- compteur : Un nom pour une variable de type entier ou caractère.
- v_init: C'est la valeur initiale que prend la variable compteur.
- v_final: C'est la valeur finale que peut prendre la variable compteur.
- - Pas : C'est la valeur dont évolue la variable compteur à la fin de chaque itération. où compteur –> compteur+pas. Généralement égal à 1.

**Observations :**

- La valeur de terminaison v_final est calculée une seule fois avant l'exécution de la boucle.
- La partie (pas) est facultative et, en son absence, signifie que **pas** est 1.
- Si le pas est positif, il est ajouté à compteur jusqu'à compteur ≥ la fin. Dans le cas d'un pas négatif, il est décrémenté jusqu'à compteur ≤ la v_final.
- compteur = v_final est exécuté.
- Si v_init est supérieur à la fin et que le pas est positif, ou si v_init est inférieur à la fin et le pas est négatif, la **boucle** pour n'est pas exécutée.
- La valeur du compteur ne peut pas être modifiée à l'intérieur de la boucle.

## 4.2. Syntaxe de « for » en C

La boucle « **for »** en C est plus générale que la boucle « **pour »** dans l]algorithme. Elle est plus proche de la boucle conditionnelle « **TantQue** » que « **pour** ».

| La forme générale | Son équivalence en algorithme |
|---|---|
| `for (initialisation ; test ; itération)`<br>`{`<br>`  Bloc d'instruction`<br>`}`<br>`Le reste des instructions` | `for (c=v_init ; c<=v_final ; c++)`<br>`{`<br>`  Bloc d'instruction`<br>`}`<br>`Le reste des instructions` |

**for** est un mot réservé en C.

La première ligne de for se compose de trois parties entre parenthèses (), toutes facultatives, séparés par un point-virgule « ; ».

- Initialisation : Cette partie est exécutée une fois avant l'exécution de la boucle. Il est généralement utilisé pour attribuer une valeur initiale au compteur. Ex : i=0
- condition: une expression de type booléen. Sa valeur doit être vrai pour exécuter la boucle. Si la condition est fausse, la boucle est quittée. Il est exécuté au début de chaque itération de la boucle. Habituellement, le compteur est testé. Comme: i<10.
- Itération: Elle est exécuté à la fin de chaque itération. Il est généralement utilisé pour incrémenter ou décrémenter le compteur. Comme : i++ ou i--.

Les instructions de « **for** » en **C** sont mis entre deux {} . Ils peuvent être omis, s'ils ne contiennent qu'une seule instruction ({} facultatif). Si nous trouvons un ensemble d'instructions après **for** et que nous ne trouvons pas les deux accolades, seule la première instruction est répétée.

**Remarques**

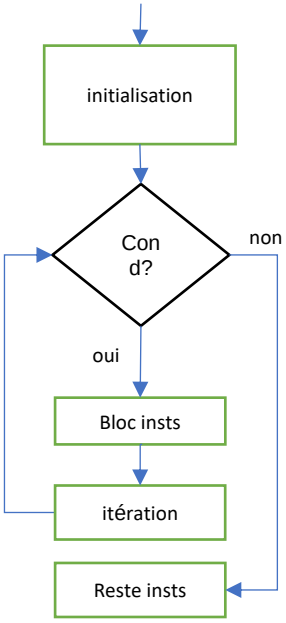
- La variable (le compteur) peut être déclarée dans la partie d'initialisation, auquel cas la portée de sa définition est uniquement à l'intérieur de la boucle for, pas à l'extérieur de celle-ci.
- La valeur du compteur dans la partie itération peut être incrémenter ou décrémenter ou modifiée de toute autre manière.
- Toutes les parties « **for** » (initialisation, test, itération) sont facultatives, elles peuvent être omises et laissées vides. mais " ; » Obligatoire et ne peut être omis. Le script suivant est valide **for** ( ; ; )
- La partie initialisation et la partie itération peuvent contenir plusieurs instructions séparées par des virgules ','.
- L'instruction « ; » est l'instruction vide.

L'exemple du code suivant est équivalent

```
int i=0;
j=10;
for ( ; ; ){
  if (!(i<j)) break;
  i++;
  j--;
}
```

```
for (int i=0,j=10 ;i<j ; i++,j--);
```

## 4.3. Algorigramme




## 4.4. Exécution :

La boucle « **pour** » s'exécute en affectant la valeur initiale au compteur, et si la valeur du compteur est ≤ de la valeur finale et que le pas est positif, le bloc d'instructions entre **faire** et **finPour** est exécuté, puis il ajoute le **pas** au compteur, puis le répète à nouveau, jusqu'à ce que le compteur devienne > de la valeur finale.
Si le pas est négatif et que la valeur du compteur est ≥ de la valeur finale, le bloc d'instructions entre faire et FinPour est exécuté, puis décrémente le pas jusqu'à ce que le compteur soit < de la valeur finale.
En C, l'expression d'initialisation n'est exécutée qu'une seule fois avant l'exécution de la boucle. La commande passe alors à la condition. Elle est testée avant chaque itération. Si le résultat est vrai, il exécute le bloc d'instructions entre {} en C, puis il exécute la partie itération, puis réévalue le test et recommence. La partie itération est exécutée à la fin de chaque itération. Lorsque le résultat du test devient faux, nous quittons la boucle en sautant aux instructions qui le suivent immédiatement.

## 4.5. Exemple :

Écrivez un programme qui lit deux nombres entiers et affiche ensuite tous les nombres entiers intermédiaires.

| Algorithme | C | ecran |
|---|---|---|
| **algorithme** nombres<br>**var** x, y, i :entier<br>/*i est le compteur*/<br>**début**<br>écrire("entrer 2 nbrs")<br>lire(x, y)<br>**pour** i←x **jusqu'à** y **Faire**<br>écrire(i)<br>**FinPour**<br>**fin** | #include <stdio.h><br>int main()<br>{<br>int x, y, i ;<br>printf("entrer 2 nbrs\n") ;<br>scanf("%d%d", &x, &y) ;<br>for **( i=x ;i<=y ;i++ )**<br>printf("%d\t", i) ;<br>return 0 ;<br>} | entrer 2 nbrs<br>5 9<br>5 6 7 8 9 |

L'algorithme prend deux nombres x et y, et a besoin d'une variable i, qui joue le rôle d'un compteur. Où il prend des valeurs successives de l'intervalle x à y. A la fin de chaque itération, 1 est ajouté au compteur i. Puisque le pas est implicitement 1. Dans le cas où x est supérieur à y, aucun nombre n'est affiché. En C, il doit être écrit.
{} a été omis dans la boucle **for** car il ne contient qu'une seule instruction.

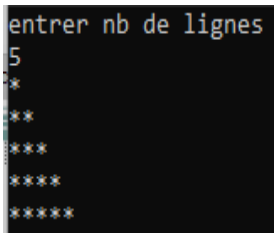
# 5. Les boucles imbriquées

Une boucle est un ensemble d'instructions répétées plusieurs fois de suite, et ces instructions peuvent être de n'importe quel type et n'importe quel nombre. Où il peut aussi s'agir d'une boucle. Dans ce cas, nous parlons de boucles imbriquées, c'est-à-dire une boucle dans une autre. Dans ce cas, elle est exécutée comme suit :

- On entre dans la boucle externe
- On entre dans la boucle interne
- La boucle interne itère jusqu'à ce qu'elle soit terminée.
- On retourne à la boucle externe pour exécuter le reste de ses instructions.
- Ainsi, nous exécutons à nouveau la boucle externe jusqu'à ce qu'elle soit terminée.

**Exemple**

Ecrire le programme qui lit le nombre de lignes n, puis affiche à l'écran dans la première ligne *, dans la deuxième **, dans la troisième ***, et ainsi de suite jusqu'à ce qu'il affiche dans la dernière ligne n *.

| Algorithme | C | الشاشة |
|---|---|---|
| **algorithme** etoile<br>**var** n, i, j :entier<br>/*i, j des compteurs*/<br>**début**<br>écrire("entrer nb de lignes")<br>lire(n)<br>**pour** i←1 **à** n **Faire**<br>**pour** j←1 **à** i **Faire**<br>écrire("*")<br>**FinPour**<br>**FinPour**<br>**fin** | #include <stdio.h><br>int main()<br>{<br>int n, i, j ;<br>printf("entrer nb de lignes\n") ;<br>scanf("%d", &n) ;<br>for **( i=1 ;i<=n ;i++ )** {<br>for **( j=1 ;j<=i ;j++ )**<br>printf("*") ;<br>printf("\n") ;<br>}<br>return 0 ;<br>} |  |

Le for(i) externe contient deux instructions : for(j) et printf("\n"). Le for(j) interne contient une seule instruction, printf("*"). printf("\n") est répété n fois. printf("*") est répété 1+2+...+n fois.

# 6. Équivalence des boucles

La boucle « **Tantque** » est utilisé lorsque nous ne connaissons pas à l'avance le nombre d'itérations et lorsqu'il est possible de ne pas exécuter le bloc d'instructions du tout.

La boucle « **Faire…Tantque** » est utilisée lorsque l'on ne connaît pas à l'avance le nombre d'itérations, et lorsque le bloc d'instructions doit être exécuté au moins une fois.

La boucle « **Pour** » est utilisée lorsque nous connaissons à l'avance le nombre d'itérations, ou lorsque nous connaissons le début et la fin de la plage du compteur.

En règle générale, toute boucle « **Tantque »** peut être exprimée par la boucle « **faire »**, en ajoutant une condition avant la boucle « **faire** ». Toute boucle « **Faire** » peut être exprimée par la boucle « **Tantque »**, en ajoutant le bloc d'instructions avant la boucle « **Tantque »**. Toute boucle « **Pour** » peut être exprimée par la boucle « **Tantque »**, en affectant la valeur initiale du compteur avant la boucle « **Tantque »**, en utilisant la valeur finale comme condition d'arrêt et en ajoutant l'instruction qui modifie la valeur du compteur à la fin de la boucle. Cependant, il n'est pas toujours possible d'exprimer la boucle « **Tantque »** ou la boucle « **faire** » avec la boucle « **pour** », à moins qu'il y ait un compteur.

En **C**, « **while »** ou « **do…while »** peuvent être exprimés avec « **for** », et toutes les boucles peuvent être exprimées avec « **goto »** et « **if »**.

**Exemples :**

**while**

| while | do…while | for | goto +if |
|---|---|---|---|

```
…
r=x ;
while ( r>y ) {
    r-=y ;
    q++ ;
}
printf(…) ;
}
```

```
…
r=x ;
if ( r>y )
  do {
    r-=y ;
    q++ ;
  } while ( r>y )
printf(…) ;
}
```

```
…
r=x ;
for ( ;r>y; ) {
    r-=y ;
    q++ ;
}
printf(…) ;
}
```

```
…
r=x ;
again :
if ( r>y ) {
    r-=y ;
    q++ ;
    goto again ;
}
printf(…) ;
}
```

## do…while

| do…while | while | for | goto +if |
|---|---|---|---|
| …<br>nb=0 ;<br>**do {**<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**} while ( x!=0 ) ;**<br>printf(…) ;<br>} | …<br>nb=0 ;<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**while ( x!=0 ) {**<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**}**<br>printf(…) ;<br>} | …<br>nb=0 ;<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**for ( ;x!=0 ; ) {**<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**}**<br>printf(…) ;<br>} | …<br>r=x ;<br>again :<br>printf("entrer un nbr") ;<br>scanf("%d", &x) ;<br>nb++ ;<br>**if ( r>y )**<br>**goto** again ;<br>printf(…) ;<br>} |

## for

| for | while | do…while | goto +if |
|---|---|---|---|
| …<br>**for (i=x;i<=y;i++)**<br>printf("%d\t",i);<br>… | …<br>**i=x ;**<br>**while ( i<=y ){**<br>printf("%d\t",i);<br>**i++ ;**<br>**}**<br>… | …<br>**i=x ;**<br>**if ( i<=y )**<br>**do {**<br>printf("%d\t",i);<br>**i++ ;**<br>**} while ( i<=y )**<br>… | …<br>**i=x ;**<br>again :<br>**if ( i<=y ) {**<br>printf("%d\t",i);<br>**i++ ;**<br>**goto** again ;<br>**}**<br>… |

# 7. Commandes pour terminer les boucles

Ces commandes sont utilisées à l'intérieure de la boucle pour effectuer une sortie anticipée de la boucle. Généralement lors de la vérification d'une condition. Toute boucle Ces commandes sont utilisées au milieu de la boucle pour effectuer une sortie anticipée de la boucle. Généralement lors de la vérification d'une condition spécifique. Toute boucle **for**, **while**, **do…while** peut être terminée en exécutant n'importe quelle instruction de saut telle que: **break**, **return** ou **goto** (vers une étiquette en dehors de la boucle). L'instruction **continue** ne fait que terminer l'itération en cours, pour sauter à la fin de la boucle et recommencer l'itération suivant. Ces instructions sont utilisées dans un « **if** ». Dans le cas de boucles imbriquées, **break** et **continue** sortent uniquement de la boucle interne.

**Exemple :**

| | |
|---|---|
| **for** (int i=1 ;i<10 ;i++){<br>**if**(i%3==0) continue ;<br>printf("%d\t", i) ;<br>} | **for** (int i=1 ;i<10 ;i++){<br>**if**(i%3==0) break ;<br>printf("%d\t", i) ;<br>} |
| Tous les nombres apparaîtront, en dépassant les multiples de 3<br>1 2 4 5 7 8 | La boucle s'arrête au premier multiple de 3<br>1 2 |