# Les boucles

Les « boucles » sont des instructions itératives ou répétitives , c'est-à-dire qu'elles permettent d'exécuter plusieurs fois certaines instructions d'un programme. Le programme « sort » de la boucle quand ces instructions cessent d'être exécutées.

Une boucle infinie est une boucle dont le programme ne sort (à priori) jamais.
Une boucle vide ne comporte pas d'instructions en dehors de la boucle elle-même.

Il existe 3 boucles en C : **while , do…while , for**

➔ *La boucle* **while**

Schéma général

**while ( *condition* )**
***instruction(s)***

où :

***condition*** est une expression considérée comme « vraie » si elle a une valeur différente de 0, et « fausse » si elle vaut 0

***instruction(s)*** est une instruction ou un bloc d'instructions encadré d'accolades **{** et **}**

Principe : Tant que ***condition*** est « vraie », ***instruction(s)*** est (sont) exécutée(s). Dès que ***condition*** devient fausse, le programme continue directement après ***instruction(s)*** . *On teste d'abord , on exécute ensuite*.

*Exemples*

```
j=0 ;    /* Il faut initialiser j avant de le tester */
while (j < 50) /* Pas de ; */
{

/* Tant que j est str. plus petit que 50 … */
        k = t * 78 ;     /* Affecter 78t à k */
        j++ ;                    /* et incrémenter j */

}       /* Fin de l'instruction while */
/* La boucle sera parcourue 50 fois */
```

```
while (1)       /* Ceci est une boucle infinie car 1 est
        différent de 0 */
{
        …
}
```

```
j=3 ;
while (j < 5 )
        printf ("j vaut %d\n",j) ;
        j++ ;
```

**/* Ceci est une autre boucle infinie, résultant d'une erreur de programmation : il n'y a pas d'accolades et donc la seule instruction exécutée à chaque itération de la boucle est le printf ; j n'est jamais incrémenté, il reste donc à 3 et comme 3<5, la boucle s'exécute indéfiniment */**

```
while (getchar() != '\n') ;
```

**/* Ceci est une boucle vide : en fait l'instruction exécutée à chaque itération de la boucle fait partie de la condition : c'est la macro getchar() . Cet exemple permet de vider le buffer d'entrée de tous les caractères présents jusqu'au premier caractère \n inclus */**

➔ *La boucle* **do…while**

Schéma général

```
do
        instruction(s)
while ( condition ) ;
```

où :

***condition*** est une expression considérée comme « vraie » si elle a une valeur différente de 0, et « fausse » si elle vaut 0

***instruction(s)*** est une instruction ou un bloc d'instructions encadré d'accolades **{** et **}**

Principe : ***instruction(s)*** est (sont) exécutée(s) tant que ***condition*** est « vraie ». Si ***condition*** devient fausse, le programme continue après le **while** . *On exécute d'abord , on teste ensuite* : ***instruction(s)*** sera (seront) donc exécutée(s) au moins une fois , même si ***condition*** est « fausse » dès le départ.

*Exemples*

```
j=0 ;
do
{
        k = r + 5 ;
        j++ ;
}
while ( j < 50 ) ;
/* On exécute d'abord l'affectation de k et
l'incrémentation de j, puis on teste si j est
str. inférieur à 50. */
/* La boucle est parcourue 50 fois */
```

```
do
{
        …
}
while (1) ;
/* Exemple de boucle infinie */
```

➔ *La boucle* **for**

Schéma général

**for (** ***initialisation*** **;** ***condition*** **;** ***réinitialisation*** **)**
***instruction(s)***

où :

***initialisation*** et ***réinitialisation*** sont des instructions

***condition*** est une expression considérée comme « vraie » si elle a une valeur différente de 0, et « fausse » si elle vaut 0

***instruction(s)*** est une instruction ou un bloc d'instructions encadré d'accolades **{** et **}**

Principe :
***initialisation*** est d'abord exécutée une fois pour toutes .
Puis ***condition*** est testée : si elle est « vraie », ***instruction(s)*** est (sont) exécutée(s).
Puis ***réinitialisation*** est exécutée.
Ensuite ***condition*** est de nouveau testée,
etc…

Dès que ***condition*** devient « fausse », la boucle s'arrête, et le programme continue après ***instruction(s)***.
*D'abord on initialise, puis on teste, puis on exécute, puis on réinitialise, puis on teste, …*

*Remarque* ***initialisation*** et/ou ***condition*** et/ou ***réinitialisation*** peuvent être omises. Seuls les **;** intérieurs sont obligatoires.
En particulier, si ***condition*** est omise, elle sera considérée comme toujours « vraie » et la boucle sera infinie.

*Exemples*

```
for ( j = 0 ; j < 50 ; j++ )      /* Pas de ; à la fin */
{
        i = k – 7;
        printf ("j vaut %d\n",j) ;
}
/* Au départ, j vaut 0 */
/* j est-il str. inférieur à 50 ? */
/* ➔  Si oui, affecter i et exécuter le printf */
/* Incrémenter j */
/* j est-il str. inférieur à 50 ? */
/* etc… */
/* ➔  Si non, ne rien exécuter et
sortir de la boucle */
/* Cette boucle sera parcourue 50 fois */
```

```
for ( j = 0 , k = 2 ; j < 50 && k <= 16 ; j++ , k = j + 2 )
{
        printf ("j vaut %d ; k vaut %d\n", j , k);
}
/* L'opérateur , permet de fusionner les 2 instructions j=0 et k=2 en une
seule. (De même pour j++ et k=j+2) */
/* La boucle démarre avec j valant 0 et k valant 2 */
/* Elle s'achèvera dès que j aura atteint ou dépassé 50 ou que k aura
dépassé 16 */
/* On vérifie aisément que cette boucle est parcourue 15 fois : k augmente
de 1 à chaque passage puisqu'il en est de même pour j ; la condition k>16
sera vérifiée avant j>=50 : on aura donc 16-2+1=15 itérations */
```

*résultat :*

```
j vaut 0 ; k vaut 2
j vaut 1 ; k vaut 3
…
j vaut 14 ; k vaut 16
```

```
for (;;)
{
        …
}
/* Ceci est une boucle infinie */
```

```
for (j=0 ; j<50 ;)
{
        …
}
/* Ceci est une autre boucle infinie, puisque j n'est pas incrémenté (a
moins de le faire dans le bloc d'instructions) */
```

```
for (i=1 ; i<=650000 ; i++) ;
/* Ceci est une boucle vide appelée "boucle de temporisation" : elle sert a
provoquer une pause ou un ralentissement du programme, le temps que
les 650000 itérations s'effectuent. A noter que dans le cas d'un float :
float f; for(f=1.0 ; f<=650000.0 ; f++);
le comptage dure plus longtemps */
```

➔ *Equivalence des boucles* **while** *et* **for**

| | |
|---|---|
| `j=0;`<br>`while ( j < 50 )`<br>`{`<br>`  k=j+2;`<br>`  j++;`<br>`}` | `for(j=0 ; j < 50 ; j++)`<br>`{`<br>`  k=j+2;`<br>`}` |