# Chapitre 4

# Structure de boucle : for

Les structures de boucles (ou structures répétitives) constituent un élément important de la programmation : elles permettent d'exécuter des instructions en boucles soit pour un nombre d'itérations fixé à l'avance, soit jusqu'à ce qu'une condition soit remplie.

## 1 La boucle POUR : `for`

Cette première forme permet de construire une boucle sur une portion de code (autour d'un bloc d'instructions) lorsque l'on connaît dès l'entrée dans la boucle le nombre d'itérations souhaitées. Sa construction s'articule autour de trois éléments :

1. un élément d'initialisation exécuté **avant toutes** les boucles.
2. un élément de condition de boucle vérifié **avant chaque** boucle.
3. un élément d'instruction de fin de boucle (ouvent une incrémentation ou une décrémentation) exécuté **après chaque** boucle.

Ces trois éléments sont séparés par des points-virgules " ;".

Syntaxe :

```
for (initialisation ; condition ; incrémentation) {
    instructions à répéter
}
```

Exemple :

```
for (i = 0 ; i < 10 ; i = i + 1) {
   printf ("iteration %d\", i) ;
}
```

Explications :

1. l'élément `i = 0` initialise la variable `i` à 0. Il suppose l'existence préalable de la variable $i$ (donc sa déclaration en amont).
2. l'élément `i < 10` constitue la condition pour entrer dans la boucle (c'est la condition qui autorise l'exécution du bloc d'instructions de la boucle).
3. l'élément `i = i + 1` correspond à l'incrémentation (on remplace parfois en `i++` à la fin de chaque boucle).

**Remarques** :

- à partir des éléments 2 et 3, on déduit ici que la boucle s'exécutera 10 fois (la variable `i` démarre avec la valeur d'initialisation 0, est incrémentée de 1 à chaque itération et la boucle s'arrête lorsque `i` vaut 10). **Dans** la boucle, la variable `i` prendra successivement toutes les valeurs comprises entre 0 et 9 (inclus). Lors de la dernière itération, `i` sera incrémenté de 1 pour atteindre 10 : c'est la condition de sortie et, après toutes les itérations, `i` vaudra donc 10.
- ces trois éléments devraient être systématiquement présents (même s'il est possible de procéder différemment) dans tous les cas usuels qui nous intéressent dans le cadre de ce module d'introduction à l'informatique.
- lorsque le bloc d'instructions n'est constitué que d'une seule instruction, les accolades deviennent optionnelle (et peuvent donc dans ce cas précis être omises).

– du fait de sa construction, une boucle `for` peut éventuellement n'être jamais [1] exécutée (si la condition d'entrée dans la boucle est systématiquement fausse). De même il faut être vigilant à ne pas créer de boucle infinie (programme qui ne s'arrête jamais) du fait soit d'une mauvaise initialisation, soit d'une condition erronée, soit d'une incrémentation absente ou mal formée.

Exemple simple :

```
/* Programme pour tester la structure "for" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main() {
    int i ;
    for (i = 0 ; i < 10 ; i = i + 1) {
        printf ("iteration %d \n", i) ;
    }
    printf ("valeur de i apres la boucle : %d \n", i) ;
    return 0 ;
}
```

## 2 Exercices

Rappel : pour afficher une valeur numérique avec printf, on peut utiliser les modificateurs de format pour ajuster la forme ou la longueur de l'affichage (exemple : %3d, %03d).

### Question 4-1 Vérification des notions de base → exercice de cours

1. Reprendre l'exemple du cours (section 1) et le tester. Programme attendu :

```
/* Programme pour tester la structure "for" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main() {
    int i ;
    for (i = 0 ; i < 10 ; i = i + 1) {
        printf("iteration %d \n", i) ;
    }
    printf ("valeur de i apres la boucle : %d \n", i) ;
    return 0 ;
}
```

2. Modifiez le programme afin de faire 100 itérations au lieu de 10. Testez.

1. même si la finalité d'une boucle `for` ne devrait pas être celle-ci : la boucle `for` doit être réservées aux situations ou le nombre d'itérations est connu à l'avance.

### Question 4-2 Liste de nombres en ordre décroissant → exercice d'assimilation

Afficher tous les nombres compris entre 100 et 0 par ordre **décroissant**. Programme attendu :

```
#include <stdio.h>

int main () {
    int i ;

    for (i = 100 ; i >= 0 ; i = i - 1) {
        printf ("iteration %d\n", i) ;
    }
    printf ("Valeur de i apres la boucle : %d\n", i) ;
    return 0 ;
}
```

### Question 4-3 Devine un nombre → exercice d'entrainement

Le but de ce jeu est de faire deviner le nombre de votre choix à un joueur qui ne le connait pas (par exemple le camarade assis à coté de vous).

1. Vous initialiserez un nombre "secret" (entre 1 et 100) avec une valeur que vous coderez "en dur" dans votre programme.
2. Votre programme demandera ensuite au joueur de deviner ce nombre. Il a 10 tentatives.
3. A chaque tentative, le programme devra dire "plus grand" ou "plus petit" ou "gagné !".

Programme attendu :

```
#include <stdio.h>

int main () {
    int secret = 19 ;
    int tentative = 10 ;
    int devine ;
    printf ("Le concepteur du programme a code un nombre secret entre 0 et 100\n") ;

    for (tentative = 10 ; tentative >= 0 ; tentative --) {
        printf ("Devinez le nombre (%d tentatives restantes)\n : ", tentative) ;
        scanf ("%d", &devine) ;

        if (devine < secret) {
            printf ("Le nombre secret est plus grand !\n") ;
        }
        else if (devine > secret) {
            printf ("Le nombre secret est plus petit !\n") ;
        }
        if (devine == secret) {
            printf ("Vous avez gagne !\n") ;
        }
    }
    return 0 ;
}
```

### Question 4-4 Liste de nombres pairs → exercice d'entrainement

Il s'agit d'écrire deux programmes - chacun avec un **algorithme** différent- qui affichent tous les nombres pairs inférieurs ou égal 100 :

1. le premier en utilisant l'opérateur modulo ("%") et un test. Programme attendu :

```c
#include <stdio.h>

int main () {
    int i ;
    for (i = 0 ; i <= 100 ; i = i + 1) {
        if ((i % 2) == 0) {
            printf ("%d\n", i) ;
        }
    }
    return 0 ;
}
```

2. le second en utilisant un pas d'incrémentation de 2 (attention à la valeur d'initialisation de la variable de boucle). Programme attendu :

```c
#include <stdio.h>

int main () {
    int i ;
    for (i = 0 ; i <= 100 ; i = i + 2) {
            printf ("%d\n", i) ;
    }
    return 0 ;
}
```

### Question 4-5 Élévation à la puissance → exercice d'entrainement

Soit deux entiers x et n ($n \geq 0$). Calculer $x^n$ par multiplications successives (sans utiliser la fonction "élévation à la puissance" Programme attendu :

```c
#include <stdio.h>

int main () {
    int i ;
    int x ;
    int n ;
    int xi ;

    x = 2 ;
    n = 12 ;
    xi = 1 ;

    printf ("%d^%d = %d\n", x, 0, xi) ;
    for (i = 1 ; i <= n ; i ++) {
        xi = xi * x;
        printf ("%d^%d = %d\n", x, i, xi) ;
    }
    return 0 ;
}
```

### Question 4-6 Suite des nombres de Lucas → exercice d'entrainement

La suite des nombres de Lucas[2] $U_n$ est définie par la relation de récurrence $U_{n+2} = U_{n+1} + U_n$ en prenant $U_0 = 2$ et $U_1 = 1$. Calculer et afficher les 20 premiers termes de cette suite.

2. plus précisément, la suite de Lucas pour laquelle P=1 et Q=-1

Affichage attendu :

```
U0 = 2
U1 = 1
U2 = 3
U3 = 4
U4 = 7
U5 = 11
U6 = 18
U7 = 29
U8 = 47
U9 = 76
U10 = 123
U11 = 199
U12 = 322
U13 = 521
U14 = 843
U15 = 1364
U16 = 2207
U17 = 3571
U18 = 5778
U19 = 9349
U20 = 15127
```

Programme attendu :

```
#include <stdio.h>

int main () {
    int i ;
    int un2 ;
    int un1 ;
    int un ;
    int n ;

    un1 = 1 ;
    un = 2 ;
    n = 20 ;


    printf ("U0 = %d\n", un) ;
    printf ("U1 = %d\n", un1) ;

    for (i = 2 ; i <= n ; i ++) {
        un2 = un1 + un ;
        printf ("U%d = %d\n", i, un2) ;
        un = un1 ;
        un1 = un2 ;
    }
    return 0 ;
}
```

### Question 4-7 Un rectangle d'étoiles → exercice d'assimilation

Ecrire un programme permettant de prendre un nombre $l$ de lignes et un nombre $c$ de colonnes, puis de réaliser un "rectangle d'étoiles" de $l$ lignes par $c$ colonnes.
Par exemple, pour $l = 5$ et $c = 10$, le programme affichera :

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

Programme attendu :

```c
/* Exemple pour tester "scanf" */
#include <stdio.h>

int main () {
    int l ; // nombre de lignes
    int c ; // nombre de colonnes
    int i ; // pour parcourir les lignes
    int j ; // pour parcourir les colonnes

    printf("Saisissez un nombre de lignes : ") ;
    scanf("%d",&l) ;
    printf("Saisissez un nombre de colonnes : ") ;
    scanf("%d",&c) ;

    for (i = 0 ; i < l ; i ++) {
        for (j = 0 ; j < c ; j ++) {
            printf ("* ") ;
        }
        printf ("\n") ;
    }
    return 0 ;
}
```

### Question 4-8 Un triangle d'étoiles

→ exercice d'assimilation

Ecrire un programme permettant de prendre un nombre $l$ de lignes, puis de réaliser un "triangle d'étoiles" de $l$ lignes. Par exemple, pour $l = 5$, le programme affichera :

```
*
* *
* * *
* * * *
```

Programme attendu :

```
/* Exemple pour tester "scanf" */
#include <stdio.h>

int main () {
    int l ; // nombre de lignes
    int i ; // pour parcourir les lignes
    int j ; // pour parcourir les colonnes

    printf("Saisissez un nombre de lignes : ") ;
    scanf("%d",&l) ;

    for (i = 0 ; i < l ; i ++) {
        for (j = 0 ; j < i ; j ++) {
            printf ("* ") ;
        }
        printf ("\n") ;
    }
    return 0 ;
}
```

### Question 4-9 Un cadre d'étoiles

→ exercice d'entrainement

Ecrire un programme permettant de prendre un nombre $l$ de lignes et un nombre $c$ de colonnes, puis de réaliser un "cadre d'étoiles" de $l$ lignes par $c$ colonnes.
Par exemple, pour $l = 5$ et $c = 10$, le programme affichera :

```
* * * * * * * * * *
*                 *
*                 *
*                 *
* * * * * * * * * *
```

Programme attendu :

```
/* Exemple pour tester "scanf" */
#include <stdio.h>

int main()
{
    int l;            // nombre de lignes
    int c;            // nombre de colonnes
    int i;            // pour parcourir les lignes
    int j;            // pour parcourir les colonnes

    printf("Saisissez un nombre de lignes : ");
    scanf("%d", &l);
    printf("Saisissez un nombre de colonnes : ");
    scanf("%d", &c);

    for (i = 0; i < l; i++) {

        if ((i == 0) || (i == (l - 1))) {
            /* Pour tracer les 2 barres horizontales du haut et du bas */
            for (j = 0; j < c; j++) {
                printf("* ");
            }
        }
        else {
            /* Sur les lignes intermÃ©diaires, ne tracer que les extremites
               du cadre */
            for (j = 0; j < c; j++) {
                if ((j == 0) || (j == (c - 1))) {
                    printf("* ");
                }
                else {
                    printf("  ");
                }
            }
        }
        printf("\n");
    }
    return 0;
}
```

**Question 4-10 Table de multiplications** → pour aller plus loin

Afficher la table de multiplications afin d'obtenir un résultat similaire à la capture d'écran suivante. (il faut utiliser 2 boucles imbriquées).

Affichage attendu :

```
x*y  |1   2   3   4   5   6   7   8   9   10
-------------------------------------------------------------------------------
1    |1   2   3   4   5   6   7   8   9   10
2    |2   4   6   8   10  12  14  16  18  20
3    |3   6   9   12  15  18  21  24  27  30
4    |4   8   12  16  20  24  28  32  36  40
5    |5   10  15  20  25  30  35  40  45  50
6    |6   12  18  24  30  36  42  48  54  60
7    |7   14  21  28  35  42  49  56  63  70
8    |8   16  24  32  40  48  56  64  72  80
9    |9   18  27  36  45  54  63  72  81  90
10   |10  20  30  40  50  60  70  80  90  100
```

**Note :** pour bien aligner vos colonnes, vous pouvez utiliser le caractère de tabulation (TAB) qui s'utilise dans le `printf` à l'aide de la séquence `\t` Programme attendu :

```c
#include <stdio.h>

/* Affiche la table de multiplication */
int main () {
    int i ;
    int j ;
    int taille ;

    taille = 10 ;

    /* la premiere ligne */
    printf ("x*y    |") ;
    for (i = 1 ; i <= taille ; i ++) {
        printf ("%d    ", i) ;
    }
    printf ("\n") ;

    /* la seconde ligne (les pointilles) */
    for (i = 1 ; i <= taille ; i ++) {
        printf ("--------") ;
    }
    printf ("\n") ;

    /* les autres lignes */
    for (i = 1 ; i <= taille ; i ++) {
        printf ("%d    |", i) ;
        for (j = 1 ; j <= taille ; j ++) {
            printf ("%d    ", i*j) ;
        }
        printf ("\n") ;
    }
    return 0 ;
}
```

### Question 4-11 Bases 10, 8 et 16 → pour aller plus loin

Afficher les nombres compris entre 0 et n saisi par l'utilisateur (sans dépasser 20) en décimal, en octal et en hexadécimal. Affichage attendu pour $n = 20$ :

```
dec     oct     hex
0       0       0
1       1       1
2       2       2
3       3       3
4       4       4
5       5       5
6       6       6
7       7       7
8       10      8
9       11      9
10      12      a
11      13      b
12      14      c
13      15      d
14      16      e
15      17      f
16      20      10
17      21      11
18      22      12
19      23      13
20      24      14
```

Note : pensez à utiliser les différents formats de la fonction `printf` (cf. page 10) pour réaliser les conversions octales et hexadécimale.

Programme attendu :

```
#include <stdio.h>
#include <math.h>

int main () {
    int n ;
    int i ;
    int resultat ;

    printf ("? nombre n : ") ;
    scanf ("%d", &n) ;

    if (n > 20) {
        printf ("Erreur : saisir un nombre inferieur ou egal a 20.\n") ;
        return 1 ;
    }

    printf ("dec \t oct \t hex\n") ;
    for (i = 0 ; i <= n ; i ++) {
        printf ("%d \t %o \t %x\n", i, i, i) ;
    }

    return 0 ;
}
```

## 3 Validation des compétences acquises à l'issue de cette séance

Je maitrise les compétences demandées à l'issue de cette séance si **je suis capable** de :

- ☐ réaliser une série d'instruction exactement $n$ fois à l'aide d'une boucle `for`
- ☐ utiliser la variable de boucle de la boucle `for` à l'intérieur des instructions de ma boucle
- ☐ imbriquer une boucle `for` dans une autre boucle `for` pour réaliser $n \times m$ séries d'instructions