Chapitre 10

Autres boucles

Ce chapitre présente d'autres types de boucles que le `while` et montre différents exemples d'imbrication de boucles.

10.1 De nouvelles boucles

10.1.1 La boucle `Tant que` (Rappel)

Cette première boucle, aussi appelée boucle `while`, a fait l'objet d'un chapitre entier. Nous ne rappellerons ici que les principes généraux de son fonctionnement. La syntaxe d'utilisation de la boucle `while` est la suivante :

```
1 while ( cond )
2 {
3   instruction  B1;
4   instruction  B2;
5   ...
6 }
7  instruction  Mi ;
```

On rappelle son fonctionnement séquentiel : on teste la condition `cond` (ligne 1),

1. si la condition est vérifiée, on exécute le corps de la boucle, c'est-à-dire les instructions contenues entre l'accolade ouvrante suivant le `while` et l'accolade fermante correspondante (instructions des lignes 3 à 5 dans l'exemple), puis on revient à la ligne 1.
2. si la condition n'est pas vérifiée, on n'exécute pas le corps de la boucle et on passe à la suite (ligne 7).

La condition `cond` est donc une condition d'*entrée* et de *maintien* dans la boucle.

10.1.2 La boucle `Répéter Tant que`

Cette seconde boucle, aussi appelée boucle `do-while`, est assez similaire à la bouche `while`. Sa syntaxe d'utilisation est la suivante :

```
1 do
2 {
3   instruction  B1 ;
4   instruction  B2 ;
5   ...
6 }
7 while ( cond ) ;
8  instruction  Mi ;
```

Son fonctionnement séquentiel est le suivant :

1. on exécute les instructions du corps de la boucle (lignes 3 à 5), puis arrivé à la fin du corps de la boucle,
2. on évalue la condition `cond` (ligne 7)
   (a) si la condition est vérifiée, on revient à la ligne 3. (on boucle),
   (b) si la condition n'est pas vérifiée, on sort de la boucle et l'on exécute l'instruction suivante (ligne 8).

À la différence de la boucle `while`, la condition `cond` de maintien dans la boucle est évaluée après l'exécution du corps de la boucle. En conséquence, le corps de la boucle est *toujours* exécuté au moins une fois. À la différence de la condition de la boucle `while` qui était une condition d'*entrée* et de *maintien* dans la boucle, celle de la boucle `do-while` est une condition de *maintien*.

10.1.3 La boucle `Pour`

Cette troisième boucle, aussi appelée boucle `for`, comporte une *instruction d'initialisation*, une *condition* et une *instruction d'incrémentation*, ainsi qu'un bloc d'instructions associé. La boucle `for` commence systématiquement par effectuer l'instruction d'initialisation. Elle évalue ensuite la condition, et tant que celle-ci est évaluée à `true`, exécute à chaque itération le bloc d'instructions puis l'instruction d'incrémentation. La boucle `for` se termine lorsque la condition est évaluée à `false`.

Dans la pratique, la boucle `for` est utilisée lorsque l'on souhaite répéter un bloc d'instructions un nombre de fois connu à l'avance (avant d'entrer dans la boucle). Par exemple, pour parcourir les cases d'un tableau dont on connait la longueur. On utilise pour cela un itérateur numérique entier que l'on initialise avec l'instruction d'initialisation et que l'on fait varier à chaque itération à l'aide de l'instruction d'incrémentation jusqu'à une valeur limite (condition).

Voici sa syntaxe d'utilisation :

```
1 for (ini; cond; incr )
2 {
3   instruction  B1 ;
4   instruction  B2 ;
5 }
```

Son fonctionnement séquentiel est le suivant :

1. l'instruction d'initialisation de l'itérateur numérique `ini` est exécutée une seule fois, lors de l'entrée dans la boucle,
2. la condition `cond` est évaluée.
   (a) Si la condition est remplie,
      i. les instructions de la boucle sont exécutées,
      ii. l'instruction d'incrémentation de l'itérateur `incr` est exécutée,
      iii. On retourne au début de l'étape 2 (évaluation de la condition).
   (b) sinon, on *"sort"* de la boucle.

On notera que les différents éléments entre parenthèses ne sont pas évalués au même moment :

— `ini` est exécutée *une seule fois* au début,
— `cond` est évaluée à chaque tour *avant* le corps de la boucle,
— `incr` est exécutée à chaque tour *après* le corps de la boucle.

Ils sont regroupés au début pour faciliter la lecture et la définition de la boucle. Par exemple :

```
1 for (int i = 0; i < 4 ; i++ )
2 {
3   affichage (i,'\n');
4 }
```

se lit comme étant "une boucle affichant tous les entiers `i`, pour i variant de 0 à 4 exclu par pas de 1". L'affichage produit est le suivant :

```
0
1
2
3
```

Initialisation `ini`

On notera que l'itérateur `i` est ici déclaré *"dans"* la boucle `for (int i ...)`. Il s'agit d'une sécurité qui limite l'utilisation de `i` au corps de la boucle (*i.e.* sortie de la boucle, la variable `i` n'est plus définie). Ainsi, le programme suivant n'est pas valide :

```
int main()
{
  for (int i = 0; i < 4 ; i++ )
  {
     affichage(i,'\n');
  }
   affichage(i,'\n');
  return 0;
}
```

Si l'on souhaite utiliser la valeur de `i` hors de la boucle, il faut déclarer la variable hors de la boucle. Le programme suivant est valide :

```
int main()
{
  int i ;
  for (i = 0; i < 4 ; i++ )
  {
     affichage(i,'\n');
  }
   affichage(i,'\n');
  return 0;
}
```

et l'affichage produit est le suivant :

```
0
1
2
3
4
```

La valeur de l'itérateur `i` en sortie de boucle est 4, ce qui est cohérent avec le fait que l'incrémentation a lieu après l'évaluation du bloc d'instructions du corps de la boucle et avant le test.

Le test `cond`

La nature du test porte généralement sur la variable entière correspondant à l'itérateur. On teste souvent si la variable est inférieure ou supérieure à une borne (la limite).

L'incrémentation `incr`

La portion paramétrant la variation de l'itérateur peut également prendre plusieurs formes. Il est par exemple possible d'opérer des décréments plutôt que des incréments. Ainsi, la boucle suivante est parfaitement correcte :

```
for (int i = 5; i > 0 ; i-- )
{
  affichage(i,'\n');
}
```

Elle affichera :

```
5
4
3
2
1
```

D'autre part, il possible de d'incrémenter par des pas différents de 1. Par exemple, en ajoutant (ou retirant 3) à chaque tour :

```
for (int i = 0; i < 10 ; i += 3 )
{
  affichage(i,'\n');
}
```

Ce qui affichera :

```
0
3
6
9
```

Remplacer les littéraux par des variables

Comme à chaque fois dans un algorithme, il sera toujours possible de remplacer un littéral par le nom d'une variable. Le programme sera correct si la variable est déclarée et initialisée avant son appel. Ainsi, le programme suivant est correct :

```
int min = 0;
int max = 10 ;
int pas = 3 ;
for (int i = min; i < max ; i += pas )
{
  affichage(i,'\n');
}
```

Attention : Sauf dans de très rares cas, il ne faut jamais modifier l'itérateur dans le corps de la boucle `for`.

▶ Sur ce thème : **Exercice 1 du TD 10**

10.2 Les boucles imbriquées

10.2.1 Qu'est-ce que des boucles imbriquées ?

On dit que deux boucles sont *imbriquées* si l'une est contenue dans le bloc d'instructions de l'autre. Certains algorithmes nécessitent d'avoir une imbrication de boucles. C'est le cas si l'on

souhaite par exemple afficher toutes les tables de multiplications de 1 à 10. En effet, l'affichage de la table de multiplication d'un nombre, par exemple la table de multiplication de 7, nécessite une boucle. De plus, il faut afficher la table de multiplication pour chaque nombre entre 1 et 10, ce qui nécessite une deuxième boucle contenant la première.

10.2.2 Premier exemple

Il s'agit d'afficher les tables de multiplication pour tous les nombres entre 1 et 10 inclus. Chaque table se présentera comme suit :

```
TABLE de 4
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
```

Ici encore, plutôt que d'essayer d'écrire directement le détail du programme, il peut être préférable de procéder par étape. Par exemple, nous pouvons dire que, globalement, notre programme doit écrire les 10 tables de multiplication de 1 à 10 et qu'il doit donc se présenter ainsi :

```
for  (int i=1; i<=10; i++)
{
  // écrire la table de i
}
```

Pour écrire la table du nombre `i`, nous pouvons procéder ainsi :

```
affichage ("\n\nTABLE de ", i, '\n');
for (int j=1; j<=10; j++) //Pour chaque nombre j variant de 1 à 10
  affichage (i, " * ", j, " = ", i*j, '\n');
```

En insérant dans la boucle `for` principale le code permettant l'écriture de la table de multiplication, nous obtenons le code complet suivant :

```
for  (int i=1; i<=10; i++)
{
  affichage ("\n\nTABLE de ", i, '\n');
  for (int j=1; j<=10; j++) //Pour chaque nombre j variant de 1 à 10
    affichage (i, " * ", j, " = ", i*j, '\n');
}
```

Remarque : On peut remplacer l'écriture de boucles imbriquées par l'appel de fonction. Par exemple, si l'on définit la fonction `tableMult` permettant d'écrire la table de multiplication d'un nombre `nb`, l'algorithme devient :

```
for(int i = 1 ; i <= 10 ; i++)
        tableMult(i);
```

La fonction `tableMult` contenant une boucle, il y a bien imbrication de boucles pour l'ordinateur mais le fait de transférer la boucle correspondant à l'écriture de la table de multiplication de l'entier `i` dans la fonction `tableMult` permet, lors de l'écriture de l'algorithme, de n'écrire qu'une simple boucle.

▶ Sur ce thème : **Exercice 2 du TD 10**

10.2.3 Deuxième exemple

On souhaite écrire un programme permettant de jouer au nombre magique. L'ordinateur choisit un nombre aléatoire entre 1 et 100 inclus et le joueur a 10 tentatives pour déterminer ce nombre. À chaque tentative, si le nombre saisi par le joueur n'est pas le nombre aléatoire choisi par l'ordinateur, le programme indique si le nombre du joueur est plus petit ou plus grand que le nombre aléatoire. On souhaite que le programme permette de jouer plusieurs fois de suite à ce jeu. Ainsi, après chaque partie, l'ordinateur demande à l'utilisateur s'il veut rejouer. Le programme s'arrête dès que le joueur ne veut plus jouer.

Pour réaliser un tel programme, il faut d'abord déterminer le code permettant de jouer au nombre magique. On a donc un nombre choisi aléatoirement et une boucle qui demande à l'utilisateur de saisir un nombre puis indiquant si celui-ci est plus petit ou plus grand que le nombre magique, et ce jusqu'à ce que l'utilisateur ait effectué 10 tentatives ou trouvé le nombre magique.

Un exemple de code est le suivant.

```
int nombreAleatoire = 1 + rand()%100;
int tentative = 0;

do
{
  tentative ++;
  affichage("Saisissez un nombre : ");
  saisie(nombre);
  if (nombre < nombreAleatoire)
    affichage("Le nombre magique est plus grand\n");
  else if (nombre > nombreAleatoire)
    affichage("Le nombre magique est plus petit\n");

}
while (nombre != nombreAleatoire && tentative < 10);


if ( nombre==nombreAleatoire)
  affichage("Vous avez gagné\n");
else
  affichage("Vous avez perdu\n");
```

Ce code doit être exécuté une première fois. Le programme doit ensuite demander à l'utilisateur s'il souhaite rejouer une autre fois et ceci doit se répéter tant que l'utilisateur saisit la lettre `O` pour oui. Cette boucle est alors de la forme :

```
char rejoue;
do
{
  affichage("Nouvelle partie\n");

  //Code de la partie...

  affichage("Voulez-vous rejouer (O/N) ?\n");
  saisie(rejoue);
}
while (rejoue == 'O'); //Tant que l'utilisateur souhaite rejouer
```

En insérant le code permettant de jouer à une partie, on obtient alors le programme complet suivant :

```
int main ()
{
```

```
3   //Initialiser le générateur de nombres aléatoires
4   //Ne faire qu'une seule fois dans le programme!!!!
5   srand(time(NULL));
6
7   int nombreAleatoire;
8   int nombre;
9   char rejoue;
10
11  do
12  {
13     affichage ("Nouvelle partie\n");
14    nombreAleatoire = 1 + rand()%100;
15    int tentative = 0;
16
17    do
18    {
19       tentative ++;
20       affichage ("Saisissez un nombre : ");
21       saisie (nombre);
22      if (nombre < nombreAleatoire)
23        affichage ("Le nombre magique est plus grand\n");
24      else if (nombre > nombreAleatoire)
25        affichage ("Le nombre magique est plus petit\n");
26
27    }while(nombre != nombreAleatoire && tentative < 10);
28
29
30    if ( nombre==nombreAleatoire)
31       affichage ("Vous avez gagné\n");
32    else
33      affichage ("Vous avez perdu\n");
34     affichage ("Voulez-vous rejouer (O/N) ?\n");
35     saisie (rejoue);
36  }while(rejoue == 'O');
37
38  return 0;
39  }
```

Attention : Il faut faire très attention à l'initialisation de la boucle à l'intérieur. Pour chaque partie, il faut choisir un nombre aléatoire et mettre le compteur de tentatives à 0. Autrement dit, les instructions correspondantes (lignes 14 et 15) sont faites à l'intérieur de la boucle `do while` et non au début du programme !

▶ Sur ce thème : **Exercices 3, 4 à 6 du TD 10 et**

TD10 : Autres boucles

✓ Exercice 1 : Les boucles*

Question 1.1 : Proposer 3 algorithmes utilisant respectivement les boucles `while`, `do-while` et `for` permettant d'obtenir la trace suivante.

```
1
4
7
10
```

Question 1.2 : Répéter le même exercice avec cette nouvelle trace.

```
Countdown
 Saisir  la duree du comte à rebours : 6
Debut !
6
5
4
3
2
1
Mise a feu !!!!
```

® Exercice 2 : Puissance d'un circuit*

Écrire un programme qui permet d'afficher la puissance d'un circuit en fonction de la tension `u` variant par pas de 0.5V et de l'intensité `i` variant par pas de 0.05A sachant que les valeurs initiales et finales de `u` et `i` seront demandées à l'utilisateur.

Exemple d'affichage : (pour `u` variant de 2 à 3V et `i` variant de 0.15 à 0.3A)

```
u = 2, i = 0.15, p = 0.3
u = 2, i = 0.2, p = 0.4
u = 2, i = 0.25, p = 0.5
u = 2, i = 0.3, p = 0.6
u = 2.5, i = 0.15, p = 0.375
u = 2.5, i = 0.2, p = 0.5
u = 2.5, i = 0.25, p = 0.625
u = 2.5, i = 0.3, p = 0.75
u = 3, i = 0.15, p = 0.45
u = 3, i = 0.2, p = 0.6
u = 3, i = 0.25, p = 0.75
u = 3, i = 0.3, p = 0.9
```

✓ Exercice 3 : Saisie de valeurs positives dans un tableau*

Définir une fonction `saisieValPositives` prenant en paramètre un tableau d'entiers et son nombre d'éléments, et initialisant les cases du tableau avec des valeurs positives saisies par l'utilisateur. La saisie de chaque valeur sera répétée jusqu'à ce que la valeur donnée soit positive.

✓ Exercice 4 : x^y**

Écrire un programme permettant de calculer x^y avec x et y entiers positifs. Seule l'addition est autorisée.

® Exercice 5 : Nombre de façons de rendre la monnaie**

Question 5.1 : Écrire un programme qui affiche toutes les manières possibles d'obtenir 1 Euro avec des pièces de 2 centimes, 5 centimes et 10 centimes. On veillera à ne pas afficher les 0 (`20*5 = 100` et non `0*2+20*5*0*10=100`). Le programme calculera et affichera le nombre total de manières d'obtenir ainsi 1 Euro.

Question 5.2 : Modifier le programme précédent pour qu'il affiche le nombre de manières possibles et toutes ces manières d'obtenir une somme `somme` avec des pièces de 2 centimes, 5 centimes et 10 centimes. La somme (en centimes) sera saisie par l'utilisateur.

® Exercice 6 : Tri à bulles***

On souhaite trier un tableau d'entiers à l'aide de l'algorithme du tri à bulles. Cet algorithme fonctionne de la manière suivante. Pour toute case du tableau sauf la dernière, on compare la valeur de cette case avec celle de sa voisine de droite. Si sa valeur est supérieure à celle de sa voisine, on échange les deux valeurs. On recommence ceci tant que l'on a échangé au moins deux valeurs.

Question 6.1 : Écrire le programme triant un tableau d'entiers à l'aide du tri à bulles.

Question 6.2 : Découper l'algorithme précédent en fonctions.

TP10 : Autres boucles

Exercice 7 : Dessins d'étoiles*

Question 7.1 : Définir une fonction affichant un rectangle d'étoiles de taille `n` × `m`, où `n` et `m` sont deux paramètres de la fonction.

Exemple : pour `n`=3 et `m`=4

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

Question 7.2 : Définir une fonction affichant un triangle rectangle dont la hauteur et la base dépendent d'un entier `n` passé en paramètre.

Exemples :

pour `n = 1`

```
*
```

pour `n = 2`

```
*
**
```

pour `n = 3`

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

pour `n = 4`

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

Question 7.3 : Écrire une fonction `deuxTrianglesRectangles` qui prend en paramètre la hauteur et qui affiche la figure suivante. Les symboles utilisés sont des étoiles (*) et des traits (-).

Exemple : pour une hauteur de 5

```
*----
**---
***--
****-
*****
```

Question 7.4 : Écrire une fonction qui affiche un triangle isocèle dont la hauteur dépend d'un entier `n` passé en paramètre.

Exemples :

pour `n`=1

```
*
```

pour `n`=2

```
 *
***
```

pour `n`=3

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

pour `n`=4

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

Question 7.5 : Écrire une fonction qui affiche un losange dont la longueur des diagonales est un entier impair `n` passé en paramètre. Dans le cas où `n` est pair, la fonction affichera un message d'erreur.

Exemples :

pour `n`=1

```
*
```

pour `n`=3

```
 *
***
 *
```

pour `n`=5

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

pour `n`=7

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

Question 7.6 : Écrire un programme principal appelant les différentes fonctions de dessin d'étoiles

Exercice 8 : Table de multiplication*

Question 8.1 : Écrire un programme qui calcule et affiche les tables de multiplication de 1 à 10. Une exécution de ce programme donnera à l'écran ce qui suit.

```
x | 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
```

Question 8.2 : Définir une fonction `afficheAvecEspaces` qui affiche un entier `x` sur `n` espaces aligné à droite (`x` et `n` étant les deux paramètres de la fonction). À titre d'exemple, l'affichage du code

```
1 afficheAvecEspaces(10,5);
2 afficheAvecEspaces(112,5);
3 afficheAvecEspaces(9,5);
4 afficheAvecEspaces(2222,5);
```

doit être

```
1    10  112    9 2222
```

Modifier alors le code de la table de multiplication pour obtenir l'affichage suivant :

```
1  x  |   1   2   3   4   5   6   7   8   9  10
2  ------------------------------------------
3   1 |   1   2   3   4   5   6   7   8   9  10
4   2 |   2   4   6   8  10  12  14  16  18  20
5   3 |   3   6   9  12  15  18  21  24  27  30
6   4 |   4   8  12  16  20  24  28  32  36  40
7   5 |   5  10  15  20  25  30  35  40  45  50
8   6 |   6  12  18  24  30  36  42  48  54  60
9   7 |   7  14  21  28  35  42  49  56  63  70
10  8 |   8  16  24  32  40  48  56  64  72  80
11  9 |   9  18  27  36  45  54  63  72  81  90
12 10 |  10  20  30  40  50  60  70  80  90 100
13 ------------------------------------------
```

Exercice 9 : Dessin d'étoiles compressé à l'aide d'un tableau d'entiers**

Un dessin d'étoiles est un dessin contenant uniquement des espaces, le symbole `*` et les retours à la ligne. Les dessins de l'Exercice 7 excepté celui de la troisième question sont des exemples de dessin d'étoiles. Pour stocker un dessin d'étoiles de manière "compressée", on peut représenter ce dessin par un tableau d'entiers. Chaque nombre positif du tableau correspond au nombre de symboles (espace ou `*`) consécutifs dans une ligne du dessin. La valeur -1 correspond à un retour à la ligne. On suppose que chaque ligne commence par des étoiles. À titre d'exemple, le triangle isocèle pour `n=4` dans l'Exercice 7 (question 4) serait représenté par le tableau

```
int tableau[14] = {0, 3, 1, -1, 0, 2, 3, -1, 0, 1, 5, -1, 7,-1};
```

En effet, la première contient 0 étoile, suivi de 3 espaces puis d'une étoile. La deuxième ligne contient 0 étoile, 2 espaces puis 3 étoiles, etc.

Question 9.1 : Définir la fonction `afficheDessinEtoiles` prenant en paramètre un tableau d'entiers et son nombre d'éléments et affichant le dessin d'étoiles correspondant.

Question 9.2 : Appeler la fonction pour afficher le triangle isocèle de taille 4. Afficher également le dessin d'étoiles défini par le tableau suivant (utilisez le copier/coller pour déclarer le tableau dans votre programme principal).

```
int tableau[58] = {0,3,14,-1,0,2,2,12,2,-1,0,1,2,2,2,6,2,2,2,-1,2,16,2,-1,2,7,2,7,2,-1,2,
16,2,-1,0,1,2,14,2,-1,0,2,2,3,6,3,2,-1,0,3,2,10,2,-1,0,4,12,-1};
```