C#

Les boucles

SOMMAIRE

La nécessité des structures répétitives

L'itération (ou structure répétitive ou **boucle**) permet d'obtenir une action composée par la répétition d'une action élémentaire ou composée, répétition qui continue tant qu'une condition n'est pas remplie, ou cesse lorsqu'une condition donnée est remplie

Exemple : La table de multiplication par 5

Avec les instructions définies à ce stade, la seule possibilité d'écrire la table en totalité est donnée par le programme ci-dessous.

```
namespace Repetitives
{
  class Test1
  {
    static void Main()
    {
      Console.WriteLine("Table de multiplication par 5");
      Console.WriteLine("============================");
      Console.WriteLine("{0} * 5 = {1}", 1, 1 * 5);
      Console.WriteLine("{0} * 5 = {1}", 2, 2 * 5);
      Console.WriteLine("{0} * 5 = {1}", 3, 3 * 5);
      Console.WriteLine("{0} * 5 = {1}", 4, 4 * 5);
      Console.WriteLine("{0} * 5 = {1}", 5, 5 * 5);
      Console.WriteLine("{0} * 5 = {1}", 6, 6 * 5);
      Console.WriteLine("{0} * 5 = {1}", 7, 7 * 5);
      Console.WriteLine("{0} * 5 = {1}", 8, 8 * 5);
      Console.WriteLine("{0} * 5 = {1}", 9, 9 * 5);
      Console.WriteLine("{0} * 5 = {1}", 10, 10 * 5);

      // Affichage persistant
      Console.ReadLine();

    }
  }
}
```

A la lecture de ce programme, on s'aperçoit vite que la même action élémentaire (moyennant paramétrage) est répétée un certain nombre de fois.

En généralisant, on peut écrire que l'instruction suivante

```
      Console.WriteLine("{0} * 5 = {1}", i, i * 5);
```

Se répète pour une valeur de i, variant de 1 à 10, la condition d'arrêt pouvant s'énoncer

Pour **i** variant de **1** à **10**

Ou

Tant que **i <= 10**

Ou

Jusqu'à **i > 10**

On distingue généralement plusieurs types de structures répétitives.

Les structures répétitives

Structure répétitive for

Syntaxe C# :

```
for ( expression_a; expression_b; expression_c )
{
    // Instructions;
}
```

expression_a représente l'initialisation des itérateurs ;
expression_b représente la condition d'itération ;
expression_c représente l'actualisation des itérateurs ;

Dans l'exemple de la table de multiplication :

```
{
  class Test1
  {
    static void Main()
    {
      Console.WriteLine("Table de multiplication par 5");
      Console.WriteLine("============================");
      for (int i =1; i <= 10 ; i++)
      {
          Console.WriteLine("{0} * 5 = {1}", i, i * 5);
      }
      // Affichage persistant
      Console.ReadLine();

    }
  }
```

Déroulement de l'exécution :

- Lors de la première exécution de l'instruction for, **i** est initialisée à **1** ;
  A chaque exécution, la condition d'itération (i <= 10) est évaluée ; si i > 10, la boucle s'arrête, et l'instruction suivant l'accolade fermante est exécutée.
- Lorsque la condition d'itération est vraie, les instructions entre accolades sont exécutées.
- Sur l'accolade fermante, **i** est incrémenté de **1**
  Retour sur l'instruction **for**

La structure While

Syntaxe C# :

```
while ( condition )
{
    // Instructions;
}
```

condition est une expression booléenne (type **`bool`**). Les **instructions** sont exécutées plusieurs fois tant que le résultat de l'expression ***condition*** est vraie (valeur **`true`**).
La condition doit pouvoir être évaluée à la première exécution de l'instruction **while**, ce qui nécessite toujours l'initialisation de la (des) variable(s) intervenant dans la condition.
Si à la première exécution du **while**, le résultat de l'expression ***condition*** est faux (valeur **`false`**), les **instructions** ne sont jamais exécutées.

Les instructions seront donc exécutées de 0 à n fois.

Exemple : La table de multiplication par 5

```
namespace Repetitives
{
  class Test2
  {
    static void Main()
    {
      int i;
      Console.WriteLine("Table de multiplication par 5");
      Console.WriteLine("============================");
      i = 1;
      while (i <= 10)
      {
          Console.WriteLine("{0} * 5 = {1}", i, i * 5);
          i++;
      }
      // Affichage persistant
      Console.ReadLine();

    }
  }

}
```

Déroulement de l'exécution :

- A chaque exécution de l'instruction **while**, la condition d'itération (i <= 10) est évaluée ; si i > 10, la boucle s'arrête, et l'instruction suivant l'accolade fermante est exécutée.
  Lorsque la condition d'itération est vraie, les instructions entre accolades sont exécutées.
- Sur l'accolade fermante, retour sur l'instruction **while**

L'instruction **while** nécessite une attention soutenue : Sa syntaxe complète est :

```
initialisation
while ( condition )
{
    // Instructions;
    // actualisation
}
```

Dans notre example,

```
i = 1;
while (i <= 10)
{
     Console.WriteLine("{0} * 5 = {1}", i, i * 5);
     i++;
}
```

Initialisation

Actualisation,

Surtout, ne pas oublier la partie Actualisation ….

L'instruction **while** par rapport à l'instruction **for** présente l'intérêt de pouvoir évaluer une condition d'itération complexe, par exemple :

```
while ((i <= 10) && (j!= 2)){…}
```

ou

```
while (!trouve) {…}              // bool trouve
```

La structure do

Syntaxe C# :

```
do
{
    // Instructions;
} while ( condition );
```

condition est une expression booléenne (type **`bool`**). Les **Instructions** sont exécutées plusieurs fois tant que le résultat de l'expression ***condition*** est vraie (valeur **`true`**).

L'instruction **do** est toujours accompagnée d'une instruction **while**.

Elle est similaire à l'instruction **while**, sauf que l'évaluation de la condition d'itération s'effectue en fin de boucle, et non pas au début, ce qui signifie que, contrairement à l'instruction **while** qui est exécutée de **0** à **n** fois**,** une instruction **do** est exécutée **au moins une fois**.

```
namespace Repetitives
{
  class Test3
  {
    static void Main()
    {
      int i;
      Console.WriteLine("Table de multiplication par 5");
      Console.WriteLine("============================");
      i = 1;
      do
      {
          Console.WriteLine("{0} * 5 = {1}", i, i * 5);
          i++;
      } while (i <= 10);
      // Affichage persistant
      Console.ReadLine();

    }
  }
}
```

Déroulement de l'exécution :

- Les instructions entre accolades sont exécutées.
- A chaque exécution de l'instruction **while**, la condition d'itération (i <= 10) est évaluée ; si i > 10, la boucle s'arrête, et l'instruction suivant l'accolade fermante est exécutée. Lorsque la condition d'itération est vraie, retour sur l'instruction **do**

Les rupteurs

Les rupteurs, en C#, sont des instructions de branchement inconditionnel à des points stratégiques du programme.

`continue` arrêt de la boucle en cours et débranchement à l'instruction responsable de la boucle.

`break` arrêt de la boucle en cours et débranchement derrière l'accolade fermante.

`return` dans une fonction retour au programme appelant quel que soit le niveau d'imbrication des structures. La valeur **n** constitue le résultat de la fonction. Si l'instruction **`return`** est trouvée dans la fonction **`Main`**, on retourne au système d'exploitation. La valeur **n** constitue alors la valeur de retour du programme (*status*) qui peut être utilisé par le système d'exploitation (**IF ERRORLEVEL** sous DOS).

Les rupteurs sont parfois dangereux pour une structuration correcte d'un programme. Ils doivent être utilisés à bon escient. Ce sont des GOTO déguisés

Exercices

Les entiers inférieurs à N
Ecrivez un programme qui affiche les nombres inférieurs à N.

La somme des entiers inférieurs à N
Ecrivez un programme qui affiche les nombres inférieurs à N.

Somme d'un intervalle
Ecrivez un programme qui saisit deux nombres n1 et n2 et qui calcul ensuite la somme des entiers de n1 à n2.

Moyenne
Ecrire un programme qui saisit des entiers et en affiche la somme et la moyenne (on arrête la saisie avec la valeur 0)

Mini et maxi
Modifiez le programme de la moyenne pour afficher le minimum et le maximum

Multiples
Ecrire un programme qui calcule les N premiers multiples d'un nombre entier X, N et X étant entrés au clavier.
Exemple pour N=5 et X=7

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
```

Il est demandé de choisir la structure répétitive (for, while, do...while) la mieux appropriée au problème.

On ne demande pas pour le moment de gérer les débordements (overflows) dus à des demandes de calcul dépassant la capacité de la machine.

Nombre de voyelles.
Ecrire le programme qui compte le nombre de voyelles d'un mot saisi au clavier, en utilisant :

- **Length** qui rend le nombre de lettres d'une chaîne donnée.
- **Substring(p,n)** qui extrait d'une chaîne donnée une sous chaîne de **n** caractères à partir de la position **p**.
- **IndexOf(schaîne)** qui restitue le rang de la première occurrence de schaîne dans chaîne donnée (si non trouvé : -1).

Calcul du nombre de jeunes, de moyens et de vieux.
Il s'agit de dénombrer les personnes d'âge inférieur strictement à 20 ans, les personnes d'âge supérieur strictement à 40 ans et celles dont l'âge est compris entre 20 ans et 40 ans (20 ans et 40 ans y compris).
Le programme doit demander les âges successifs.
Le comptage est arrêté dès la saisie d'un centenaire. Le centenaire est compté.
Donnez le programme C# correspondant qui affiche les résultats.

Un nombre est-il premier
Ecrivez un programme qui permet de tester si un nombre est premier.

Nombre magique
Ecrire un programme qui met en œuvre le jeu du nombre magique :
L'ordinateur choisit un nombre aléatoire (à l'aide de la classe Random) et l'utilisateur doit trouver ce nombre. A chaque fois que l'utilisateur saisit une valeur, il reçoit une indication lui indiquant « plus petit » ou « plus grand ».