Cours n°: 4

Objectifs :.Les instructions répétitives ou traitements itératifs.

Difficultés : .Aucune difficulté majeure, juste de la pratique. ☺☺

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée... On appelle parfois ces structures *instructions répétitives* ou bien *itérations*.
La façon la plus commune de faire une boucle, est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions: c'est une boucle!
Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.

La syntaxe de cette expression est la suivante:

```
for (compteur; condition; modification du compteur) {
        liste d'instructions
}
```

Par exemple:

Fichier "tp41.cs"

```
using System;

public class tp41
{
  static void Main(string[] args)
  {

      for (int i=1; i<6; i++) {
      Console.WriteLine((char)i);
      }
  }
}
```

Cette boucle affiche 5 fois la valeur de *i*, c'est-à-dire 1,2,3,4,5.Elle commence à i=1, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours. Comme vous le voyez à l'écran, ce que je viens de vous expliquer ressemble à une blague…. A vous de corriger ;-)) D'autre part, C# autorise la déclaration de la variable de boucle dans l'instruction *for* elle-même!

- il faudra toujours vérifier que la boucle a bien une condition de sortie (i.e le compteur s'incrémente correctement)
- une instruction System.out.println(); dans votre boucle est un bon moyen pour vérifier la valeur du compteur pas à pas en l'affichant!
- il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle:
  - for(i=0;i<10;i++) exécute 10 fois la boucle (i de 0 à 9)
  - for(i=0;i<=10;i++) exécute 11 fois la boucle (i de 0 à 10)
  - for(i=1;i<10;i++) exécute 9 fois la boucle (i de 1 à 9)
  - for(i=1;i<=10;i++) exécute 10 fois la boucle (i de 1 à 10)

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.
La syntaxe de cette expression est la suivante:

```
while (condition réalisée) {
        liste d'instructions
}
```

Cette instruction exécute la liste d'instructions **tant que** (*while* est un mot anglais qui signifie *tant que*) la condition est réalisée.

La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur!
Exemples :

Fichier "tp42.cs"

```
using System;

public class tp42
{
  static void Main(string[] args)
  {
            const int fin=10;// ceci est une constante, mot clé const
            int somme=0;
            int i=0;
            while (i<fin){
                  somme=+i; //Pour rigoler essayez aussi somme+=i. Avez
vous une explication ?
                  i++;
            }
            Console.WriteLine("La somme vaut :"+somme);
  }
}
```

L'instruction do/while

A l'instar de la boucle while, la boucle do/while exécute une série d'instructions tant qu'une expression de contrôle est vrai. La grande différence tient au fait que la boucle while effectue la vérification avant le corps, alors que dans la boucle do/while la vérification s'effectue en fin de boucle. La conséquence est simple : avec la boucle while, le corps ne sera jamais exécuté si la condition est fausse au début de la première itération, alors qu'avec une boucle do/while vous êtes assuré que le corps sera exécuté au moins une fois. Dans l'exemple suivant, on lit une variable au clavier tant que la valeur de celle-ci n'est pas comprise dans un intervalle déterminé.

Fichier "tp43.cs"

```
using System;

public class tp43
{
  static void Main(string[] args)
  {
      int valeur;
      const int valinf=5;
      const int valsup=10;
      do
      {
      Console.WriteLine("Saisissez une valeur");
      valeur=System.Convert.ToInt32(System.Console.ReadLine());
      }
      while ((valeur<valinf)||(valeur>valsup));
  }
}
```

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.
La syntaxe de cette expression est "*continue;*" (cette instruction se place dans une boucle !), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.
Exemple: Imaginons que l'on veuille imprimer pour x allant de 1 à 10 la valeur de 1/(x-7) ... il est évident que pour x=7 il y aura une erreur. Heureusement, grâce à l'instruction *continue* il est possible de traiter cette valeur à part puis de continuer la boucle!

Fichier "tp44.cs"

```
using System;

public class tp44
{
  static void Main(string[] args)
  {
      int x=1;
      int a=0;
      while (x<=10)
      {
            if (x == 7)
            {
                  Console.WriteLine("Division par zero!");
                  x++ ;
                  continue;
            }
            a = 1/(x-7);
            Console.WriteLine(a);
            x++;
      }
  }
}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre condition que celle précisée dans l'en-tête de la boucle. L'instruction *break* permet d'arrêter une boucle (*for* ou bien *while*). Il s'agit, tout comme *continue*, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour!
Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur (x-7) s'annule, il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

Fichier "tp45.cs"

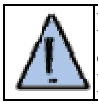
```
using System;

public class tp45
{
  static void Main(string[] args)
  {
       int x=1;
       int a=0;

       for (x=1; x<=10; x++) {
       a = x-7;
       if (a == 0) {
       Console.WriteLine("division par 0");
       break;
       }
       Console.WriteLine((char) 1/a);}
  }
}
```

L'instruction switch

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce branchement conditionnel simplifie beaucoup le test de plusieurs valeurs d'une variable, car cette opération aurait été compliquée (mais possible) avec des *if* imbriqués. Sa syntaxe est la suivante:

```
switch (Variable) {
case Valeur1:
        Liste d'instructions
        break;
case Valeur2:
        Liste d'instructions
        break;
case Valeurs...:
   Liste d'instructions
        break;
default:
        Liste d'instructions
        break;
}
```

Les parenthèses qui suivent le mot clé *switch* indiquent une expression dont la valeur est testée successivement par chacun des *case*. Lorsque l'expression testée est égale à une des valeurs suivant un *case*, la liste d'instruction qui suit celui-ci est exécuté. Le mot clé *break* indique la sortie de la structure conditionnelle. Le mot clé *default* précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs.



N'oubliez pas d'insérer des instructions break entre chaque test, ce genre d'oubli est difficile à détecter car aucune erreur n'est signalée...
L'instruction switch ne fonctionne que sur des variables entières (de type int) !

Fichier "tp46.cs"

```
using System;

public class tp46
{
  static void Main(string[] args)
  {
      for(int i=0;i<10;i++)
            {
            switch(i){
                  case 9: Console.WriteLine("Neuf"); break;
                  case 8: Console.WriteLine("Huit"); break;
                  case 7: Console.WriteLine("Sept"); break;
                  case 6: Console.WriteLine("Six "); break;
                  case 5: Console.WriteLine("Cinq"); break;
                  case 4: Console.WriteLine("Quatre");break;
                  case 3: Console.WriteLine("Trois ");break;
                  case 2: Console.WriteLine("Deux"); break;
                  case 1: Console.WriteLine("Un"); break;
                  default: Console.WriteLine("Zero");break;
                  }
            }
  }
}
```

Cet exemple vous ferra, sans aucun doute, mieux comprendre les choses. Je vous invite à le tester et à le modifier de façon à ce que vous compreniez bien toute la sémantique de cette instruction.

Exercices applicatifs

Exercice 1 :

Soit le programme suivant :

Fichier "tp4-exo1.cs"

```
using System;

public class tp46
{
  static void Main(string[] args)
  {
      int i,n,som;
      som=0;
      for (i=0;i<4;i++)
      {
            Console.WriteLine("Donnez un entier");
            n=System.Convert.ToInt32(System.Console.ReadLine());
            som+=n;
      }
      Console.WriteLine("Somme = "+som);
  }
}
```

Ecrire ce programme, puis un autre réalisant la même chose en employant une instruction while, et enfin un autre utilisant une instruction do… while.

Exercice 2

Voici un programme en C, très simple, veuillez en faire l'adaptation en C#

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  char nom[30];
  int reponse;
  char test;
  do
  {
    printf("entrez un nom : \n");
    gets(nom);
    printf("entrez un résultat \n");
    printf("1 pour admis \n2 pour admissible\n3 pour recalé \n");
    scanf("%d",&reponse);
    switch(reponse)
    {
    case 1:
       printf("Vous êtes admis %s \n",nom );
       break;
    case 2:
       printf("vous etes admissible %s \n", nom);
       break;
    default :
       printf("Vous êtes recale %s \n",nom);
       break;
    }
    printf(" \n\nPour arrêter appuyez sur z  \n");
    test=getche();
    }
  while(test!='z');
}
```