TPC#3 : Boucles + débugger

1 Structure du rendu

Voici la structure du rendu pour ce TP :

```
rendu-tp-login_x.zip/
    /login_x
        /AUTHORS            // '* login_x'
        /README             // avec comme contenu ce que vous voulez ou rien
        /Preliminaires.cs
        /Boucles.cs
        /Boucles2.cs
        /Debugger.cs
```

Vous ne devez pas rendre de code qui **ne compile pas** !

2 Introduction

Les boucles font partie des bases à connaître pour un informaticien, vous serez amené à les utiliser très régulièrement pour factoriser votre code ou pour vous déplacer dans des tableaux à n dimensions. Nous allons voir dans la première partie du tp les differentes boucles qui existent : for, while, do while.

Dans la deuxième partie du tp nous allons nous intéresser à la console de débuggage de Visual studio, il est très important de bien savoir l'utiliser car sur des gros projets vous allez sûrement passer plus de temps à corriger des bugs qu'à écrire du code.

3 Préliminaires

Pour ce tp il est primondial que vous connaissiez bien la console de visual studio, nous allons donc commencer par quelques préliminaires. Pour ceux qui connaissent déjà bien la console cette partie sera rapide à finir, pour les autres c'est l'occasion de réviser ou de vous mettre à l'aise avec la console.

Il est interdit dans les préliminaires d'utiliser **WriteLine() et ReadLine() !**

Commencez par ouvrir la solution Preliminaires.sln. Exécutez le code avec f5 ou bien la petite flèche verte en haut de votre écran.

FIGURE 1 –

Vous allez voir l'écran de console s'ouvrir puis se refermer immédiatement. Qu'est ce qui s'est passé ?
Regarder dans le code de préliminaires.cs, l'éxecution commence à partir de la ligne

```
static void Main(string[] args)
```

Ce qui s'est passé c'est que le programme a lu jusqu'à la fin de l'instruction, comme il n'y a rien il s'est ensuite refermé.

Rajoutez maintenant à l'intérieur de la fonction d'entrée (static void Main)) la ligne suivante :

```
Console.Read();
```

Cette fois ci la console s'ouvre sans se refermer. La fonction Console.Read() est ce qu'on appelle une fonction bloquante, c'est à dire que le programme s'arrête sur cette fonction jusqu'à ce qu'elle ait pu lire une donnée. Dans le cas de Read() elle attend que vous tapiez sur la touche entrée, à ce moment là elle va lire un charactère sur l'entrée de la console. Si vous tapez entrée vous verrez que la console se referme, en effet une fois le Read effectué il n'y a plus rien dans votre code.

Rajoutez un deuxieme Console.read(), puis exécutez et tapez sur entrée la console se referme.

Rajoutez une troisième Console.Read(), puis exécutez, cette fois ci la console reste ouverte, pourquoi ?

Ce qui se passe c'est que lorsque vous tapez sur la touche entrée, le programme va rajouter le charactères '\r' [1] et '\n' [2] puis exécuter le read() ;
Le premier read va donc lire le '\r' puis le deuxième read va lire le charactère '\n', arrivé au troisième read comme il n'y a plus de caractères à lire le programme se bloque en attendant que vous tapiez de nouveau sur entrée.
Relancez le programme et entrez un seul caractère puis tapez sur entrée, le programme se referme de suite car après que vous ayez tapé sur entrée le programme lira le caractère que vous avez entrée en console puis le \r , puis le \n, comme il ne reste plus de read() bloquant, le programme se termine. Pour vous en convaincre enlevez vos Read() et ajoutez les lignes suivantes :

```
int c;
c = Console.Read();
Console.Write(c);
Console.Write("\n");
c = Console.Read();
Console.Write(c);
Console.Read();
```

Relancez le programme et tapez sur entrée, le premier read va stocker \r dans la variable c, puis écrire son code ascii, la console écrit ensuite \n pour faire un retour à la ligne, le deuxième Read() va stocker le caractère \n dans la variable c puis l'afficher, finalement le dernier Read() comme il n'y a plus rien à lire va attendre et donc bloquer l'exécution jusqu'à ce que vous tapiez sur entrée. Vous verrez les chiffres 10 et 13 s'afficher correspondant respectivement à \r et \n.

Pourquoi C# met deux caractères pour indiquer une fin de ligne alors que un (\n) aurait suffi ? Je vous invite à regarder les raisons historiques sur wikipedia (c'est pas long) `http://fr.wikipedia.org/wiki/Carriage_Return_Line_Feed`

Il est important que vous compreniez bien que à chaque fois que vous tapiez sur entrée C# rajoutera automatiquement ces deux caractères à la suite de ce que vous avez éventuellement écrit dans la console, ça vous évitera des maux de crâne pour la suite.

Ecrivez maintenant un petit programme (dans Preliminaires.cs) qui doit se comporter de la manière suivante : au lancement, la fenêtre de la console s'ouvre, l'utilisateur tape exactement deux caractères puis sur entrée, le programme doit afficher la valeur ascii du **deuxième** caractère puis un retour à la ligne puis la valeur ascii du premier caractère, puis le programme attend que l'utilisateur tape sur entrée pour se refermer.

4 Boucles : les bases

Nous allons nous intéresser dans cette partie aux boucles for, while et do while. Commencez par ouvrir la solution Boucles.sln. Lancez le programme et vérifiez que vous avez bien tous les exos qui s'affichent du 1 au 7, tapez ensuite entrée pour refermer la fenêtre.

1. retour chariot (carriage return) code ascii : 10
2. retour à la ligne (line feed) code ascii : 13

4.1 Exo 1 : While

```
while (condition)
{
    instruction;
}
```

La boucle while fonctionne de la manière suivante : tant que la condition est vraie il va exécuter l'instruction.

Ecrivez maintenant en dessous de la ligne "Console.WriteLine("Exo 1") ;" un programme qui affiche les nombres 20 à 1 en faisant des retours à la ligne, ça devra ressembler comme ci dessous dans votre console.

```
Exo1
20
19
18
17
16
(...)
3
2
1
Exo2
(...)
```

Note : Attention aux boucles infinies ! N'oubliez pas de décrémenter/incrémenter la variable qui vous sert de compteur.

4.2 Exo 2 : While you shall not pass !

A partir de maintenant vous avez le droit aux fonctions Console.WriteLine et Console.ReadLine(). Console.WriteLine se comporte comme un Console.Write() sauf qu'il ajoute automatiquement à la fin de ce que vous avez écrit un \n.

Console.ReadLine se comporte comme un Read() sauf que lorsque l'utilisateur tape sur entrée il va lire l'ensemble des caractères qui se trouvaient dans la console et non un seul comme Read(), de plus il renvoie une string de ce qu'il aura lu.

Ecrivez maintenant en dessous de la ligne "Console.WriteLine("Exo 2") ;" un programme qui affiche dans la console "Please say ok", si l'utilisateur tape "ok" puis sur entrée le programme se termine sinon le programme affiche un petit dessin en ascii puis attend de nouveau que l'utilisateur tape "ok" puis entrée.

En bref votre programme ne laissera pas l'utilisateur passer à la suite tant qu'il n'aura pas tapé "ok" plus entrée.

Votre programme devra ressembler à ceci :

```
(...)
Exo2
Please say ok
(l'utilisateur tape 'toto' + entrée)
(dessin affiché)
(l'utilisateur tape 'tralala' + entrée)
(dessin affiché)
(l'utilisateur tape 'ok' + entrée)
Exo3.1
(...)
```

pour le dessin à afficher vous pouvez copier coller ce bout de code si vous n'avez pas d'inspiration.

```
Console.WriteLine("    _____________          ");
Console.WriteLine("   / \\           \\        ");
Console.WriteLine("  /___\\___________\\       ");
Console.WriteLine("  |  _ |    ___    |  \\o/ ");
Console.WriteLine("  | | ||   |___|   |   |  ");
Console.WriteLine("  |_|_||___________|   M  ");
```

4.3 Exo 3 : Do while

```
do
{
    instruction;
}while(condition);
```

Dans certains cas particuliers on peut avoir besoin d'éxécuter au moins une instruction avant de faire le test de condition. Ecrivez le programme qui en utilisant une boucle do while, demande à l'utilisateur d'écrire un message (par exemple 'ok'), puis qui affiche sur la ligne suivante 'your answer' avant de lire l'entrée de l'utilisateur. Si l'entrée est correcte ('ok' dans cette exemple), on passe à la suite du programme mais si l'entrée est incorrecte, on re demande à l'utilisateur d'entrer un mesaage.

un exemple ci dessous :

```
(...)
Exo 3
Please write 'ok'
your answer :
(l'utilisateur tape 'toto' + enter)
Please write 'ok'
your answer :
(l'utilisateur tape 'pepe' + enter)
Please write 'ok'
your answer :
(l'utilisateur tape 'ok' + enter)
Exo 4.1
(...)
```

4.4 Exo 4 : To be for or to be while, that's the question

```
for (initialisation; condition; incrémentation)
{
    instruction;
}
```

par exemple :

```
for (int i = 0; i < 10; i++)
{
    instruction;
}
```

Va exécuter 10 fois l'instruction.

La boucle for est un while caché c'est à dire que tout ce que vous écrirez avec un while, vous pourrez l'écrire avec un for, c'est ce qu'on va voir de suite.

Ecrivez en dessous de la ligne "Console.WriteLine("Exo 4");" une boucle while qui affiche 6 fois "test with while".

Exemple ci dessous :

```
(...)
Exo 4
test with while
test with while
test with while
test with while
test with while
test with while
test with while
Exo5
(...)
```

Maintenant écrivez une boucle for qui affiche 6 fois "test with for" . Exemple ci dessous :

```
(...)
Exo 4
test with while
test with while
test with while
test with while
test with while
test with while
test with while
test with for
test with for
test with for
test with for
test with for
test with for
test with for
Exo5
(...)
```

Vous remarquerez que la boucle for est plus compacte au niveau du code, ici elle ne nécessite que 4 lignes alors que son équivalente while demande 6 lignes.

Quand utiliser for et quand utiliser while ? En général on utilise for quand on connaît le nombre exact d'itérations à effectuer (traiter tous les elements d'un tableau par exemple), alors que la boucle while est plus pratique quand on ne connait pas à l'avance le nombre d'itérations à effectuer (attente d'une entrée utilisateur qui corresponde à une condition par exemple).

Avant de passer à la suite ajouter ces deux lignes pour conclure l'exo 4

```
Console.WriteLine("Tapez sur entrée pour continuer");
Console.ReadLine();
```

4.5 Exo 5 : For simple

Ecrivez en dessous de la ligne "Console.WriteLine("Exo 5");" un programme qui demande à l'utilisateur d'entrer une phrase à afficher 42 fois, une fois que l'utilisateur a tapé sa phrase plus entrée le programme affiche 43 fois le message sous le format suivant :

(le nombre de fois que la phrase a été affiché) + ' : ' + (la phrase ou le mot que l'utilisateur a tapé)

Un exemple ci dessous de ce que vous pourriez voir dans la console :

```
Exo 5
Veuillez ecrire le message à afficher 43 fois
(l'utilisateur entre 'toto' puis entrée)
1 : toto
2 : toto
3 : toto
(...)
41 : toto
42 : toto
43 : toto
Exo 6.1
(...)
```

4.6 Exo 6.1 : For double

Copiez collez le bout de code suivant en dessous de la ligne "Console.WriteLine("Exo 6.1")".

```
int[,] tab = new int[,] { { 1, 1, 1, 1 },
                          { 2, 2, 2, 2 },
                          { 3, 3, 3, 3 },
                          { 4, 4, 4, 4 } };
```

Il s'agit d'un tableau à deux dimensions, pour accéder à la première ligne, troisième colonne vous pouvez faire tab[0,2].

Vous devez utiliser ce tableau pendant tout l'exercice 6 !

Ecrivez maintenant le programme qui affiche grâce à deux boucles for imbriquées la sortie exacte suivante dans la console :

```
-----------------
| 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |
-----------------
```

Note : Pour la première et la derniere ligne avec des '- - - - - - ' vous pouvez les mettre en dehors des deux boucles for, idem pour tous les exos du 6.

4.7 Exo 6.2 : For double encore

Ecrivez à la suite le programme qui affiche grâce à deux boucles for imbriquées la sortie exacte suivante dans la console :

```
-----------------
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
-----------------
```

Indice : Vous avez très peu de choses à modifier par rapport à l'exercice précedent.

4.8 Exo 6.3 : For double encore et encore

Ecrivez à la suite le programme qui affiche grâce à deux boucles for imbriquées la sortie exacte suivante dans la console :

```
----------------------
| 42 |  1 |  1 |  1 |
|  2 | 42 |  2 |  2 |
|  3 |  3 | 42 |  3 |
|  4 |  4 |  4 | 42 |
----------------------
```

Si vous avez fini tout l'exo 6 vous pouvez ajouter les deux lignes suivantes

```
Console.WriteLine("Tapez sur la touche entrée pour continuer");
Console.ReadLine();
```

Normalement vous devriez avoir ceci comme rendu pour l'exo 6

```
Exo 6.1
-----------------
| 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |
-----------------
Exo 6.2
-----------------
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
-----------------
Exo 6.3
----------------------
| 42 |  1 |  1 |  1 |
|  2 | 42 |  2 |  2 |
|  3 |  3 | 42 |  3 |
|  4 |  4 |  4 | 42 |
----------------------
```

4.9 Exo 7 : For double+

En dessous de la ligne "Console.WriteLine("Exo 7")" écrivez le programme qui utilise deux boucles imbriquées pour calculer la somme de 1 à 10.

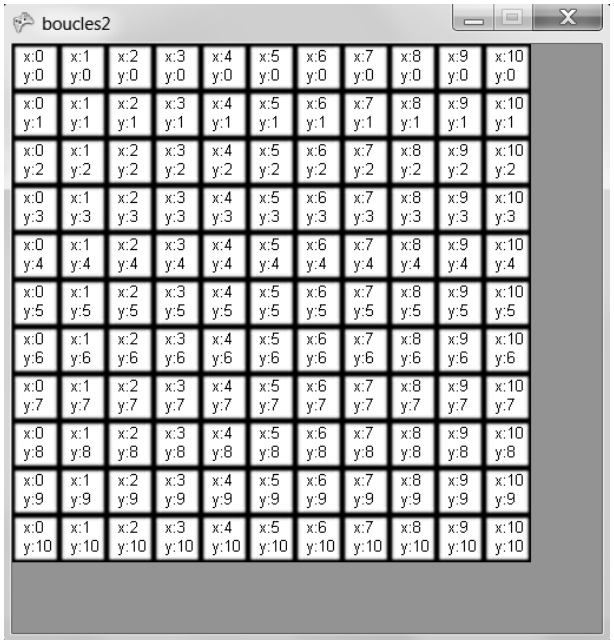
La sortie devra s'afficher **exactement** comme ceci :

```
Exo 7
1 = 1
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 + 5 + 6 = 21
1 + 2 + 3 + 4 + 5 + 6 + 7 = 28
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
```

5 Boucles2

Parfois on peut se retrouver avec un tableau ou une liste à une seule dimension mais qui nécessitent un traitement en 2D.

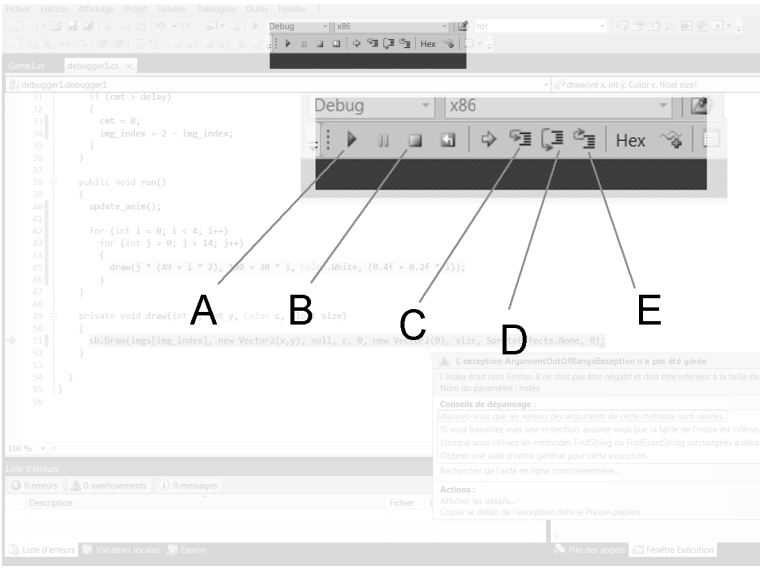
Ouvrez la solution Boucles2.sln. Essayer d'afficher l'image suivante **uniquement** en modifiant les paramètres de draw (ligne 36).



Indice : Pensez à utiliser le modulo '%' et la division entière '/' pour convertir x en une coordonnée 2d.

6 Debugger

Sur les gros projets vous allez passer beaucoup de temps à débugger, il est donc important de bien connaître l'interface et les options de debug. Ouvrez la solution debugger.sln et lancez l'exécution du programme avec f5 (ou la flèche verte). Le programme va se planter et vous remarquerez que vous disposez désormais de nouvelles options dans votre menu.



6.1 A : continuer

Cette option vous permet de demander au débugger d'ignorer l'erreur et passer à la suite, commentez la ligne en jaune puis cliquez sur cette flèche vous verrez que le programme continue de s'exécuter.

6.2 B : arrêter le débuggage

Cette option vous permet de demander au débugger de s'arrêter.

6.3 C D et E : pas à pas

Pour comprendre ces options nous allons faire appel à quelque chose qu'on appelle un point d'arrêt.

Fermer la session de débuggage en appuyant sur le bouton arrêter le débuggage (B).

Maintenant cliquez sur la partie vide de votre interface à gauche de la ligne updateAnim().

Un petit bouton rouge apparaît, c'est un point d'arrêt, quand votre programme passera par cette ligne elle s'arrêtera. Relancez l'exécution avec f5 et vous verrez que le programme s'arrête sur cette ligne. Maintenant vous pouvez avancer pas à pas avec le bouton (C), avancer pas à pas mais sans entrer dans une sous fonction (D) ou bien avancer jusqu'à la sortie de la fonction (E). Passez un peu de temps à

manipuler ces 3 options pour bien comprendre comment elles fonctionnent.

Le point d'arrêt peut être très pratique surtout si vous voulez vous assurer que le programme ne passe pas par une certaine ligne ou condition.

6.4 Exercice de débuggage

Une autre technique supplémentaire pour débugger consiste à isoler l'erreur. Vous pouvez par exemple commenter certaines lignes ou forcer certaines valeurs pour voir si l'exécution du programme fonctionne toujours ou pas.

Essayez de trouver dans le fichier debugger.cs ce qui ne va pas et corrigez le.

Indice : Il n'y a qu'une seule petite chose à changer. Si ça marche vous êtes sensés voir des trololols danser.