Practical Work - Session 3 Loops and Debugger

1 Project structure

This is how your work should be structured

```
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
```

Please do not hand in code that **does not compile**!

2 Introduction

Loops are part of the basic knowledge of a computer engineer. You will use them a lot in your code, for code factorization or to perform actions on arrays with one or more dimensions. In the first part of this session, we will see that different kinds of loops exist : "for", "while" and "do while".

In the second part, we will see the Visual Studio debugging console. It is very important to be able to use it efficiently. In big projects, you may have to put more work in bug squashing than in programming.

3 Preliminaries

For this practice session you need to know the Visual Studio console very well, which is why we will start with a few reminders about it. For those who know the console already, this part will be little work. For the others, it is a good opportunity to practice and become comfortable with it.

In the preliminaries it is forbidden to use these two functions **WriteLine() et ReadLine()!**

Start by opening the "Preliminaires.sln" solution. Run the code with the F5 key or by clicking on the little green "play" button on the top of your screen.

FIGURE 1 –

You will see a console screen pop up and then close quckly. What has happened?

Look at the code in the preliminaires.cs file. The execution flow starts at the following line :

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

When the program ran, it started going through the main function, reached the end and basically did nothing, and then closed the window.

Now, add the following line in the main function :

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

This time, the console opens but does not close. The "Console.Read()" function is a blocking function, which means that the execution will not continue until it reads some data. Once you press a key, Read() gets it's data and the execution continues, which is why the console closes.

Add a new Console.Read(), then run your program again. You will notice that the console closes when you press once on the return key.

Add a third one. This time the console will not close when you press the return key once. Why ?

When you press the return key, you send exactly two characters into the input : '\r' [1] and '\n' [2] before executing the function read() ;
The first Read() will receive the '\r' and the second one will receive the '\n', When it gets to the third one, the program stops and waits for more input from the user.
Start the program again, enter a single character and press Enter. The program closes immediately because every Read() function has got its data. The first one reads the character you entered, the second one will get the \r , and the last one the \n. Try these lines out to better understand this behaviour :

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

Launch the program and hit the return key. The first Read() will store the \r in the c variable, then print its ascii code value, and print a \n to go to a new line. The second Read() will put the \n in the c variable. We then use Write to print it. Finally, the last Read() blocks the function and waits for the user to press a key and end the program.You will see 10 and 13 written on your screen which are the ascii values of \r and \n.

Why does pressing the Enter key send two characters to end a line when we see in this example that a (\n) is enough ? You can read more information about the historical reasons for this behaviour on this page : `http://fr.wikipedia.org/wiki/Carriage_Return_Line_Feed`

Write a small program in Preliminaires.cs that has the following behaviour : when started, the console opens, the user types in two characters, then presses the return key, the program should write the ascii value of the **second** character, then a new line and then the value of the first character. The program then waits for the user to press a key to close.

4 Basic loops

We will take a look at the behaviour of "for", "while" and "do while" loops in this part. Start by opening the Boucles.sln solution. Start the program and check if you have every exercise displayed in the console. Press Enter to close the console.

4.1 Ex1 : While

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

1. carriage return ascii code : 10
2. line feed ascii code : 13

The While loop works in the following way : while the condition is true, execute the instruction.

Under "Console.WriteLine("Exo 1");" write a loop that displays every number between 20 and 1 on the console.

It should look something like this :

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

Note : Be wary of infinite loops ! If your condition is always true, you will be stuck in your while block. Do not forget to increment/decrement your counter.

4.2 Ex2 : While you shall not pass !

From now on you are allowed to use the Console.WriteLine() and Console.ReadLine() function. Console.WriteLine behaves exactly like Console.Write, but adds a new line at the end of what it writes.

Console.Readline() reads the input of a user up to a newline and returns a string. It reads more than one character, unlike Read().

Under "Console.WriteLine("Exo 2") :" write a program that writes "Please say ok" to the console, then waits for user input. If the user enters "ok" the program ends. In other cases, it prints an ascii art and writes "Please say ok" again.

This is what the output should look like :

```
(...)
Exo2
Please say ok
(user enters 'toto' + enter)
(dessin affiché)
(user enters 'tralala' + enter)
(dessin affiché)
(user enters 'ok' + enter)
Exo3.1
(...)
```

You can use this for your ascii art if you want.

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

4.3 Ex3 : Do while

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

In a few special cases we need to execute the instruction inside the loop at least once before testing the condition. Using the "do while" construction, write a program that asks the user to write something on the console, then writes "your answer :" and waits for user input. Continue until the user enters the correct answer.

An example :

```
(...)
Exo 3
Please write 'ok'
your answer :
(user enters 'toto' + enter)
Please write 'ok'
your answer :
(user enters 'pepe' + enter)
Please write 'ok'
your answer :
(user enters 'ok' + enter)
Exo 4.1
(...)
```

4.4 Ex4 : To be for or to be while, that is the question

```
for (init; condition; incrementation)
{
    instruction;
}
```

For example :

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

Will execute the instruction 10 times.

The "for" loop is a hidden "while" loop. This means that with a "while", you can achieve the same results as with a "for".

Write a program under "Console.WriteLine("Exo 4") :" that displays "test with while" 6 times.

Example :

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

Now write a loop that prints 6 times "test with for". Example :

```
(...)
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
(...)
```

You will notice that you can achieve the same results with both constructs quite easily, even if the "for" takes less space.

When should you use "for" and when should you use "while" ? Generally, you should use "for" when you execute an instruction a set number of times (processing every element of an array for example), whereas the while loop is easier to use when you don't know the number of iterations that you will do.

Add this to your code to finish point 4 and continue

```
Console.WriteLine("hit enter to continue");
Console.ReadLine();
```

4.5 Ex5 : A simple for

Under "Console.WriteLine("Exo 5") :" write a program that asks the user to enter a string that will be displayed 43 times. The following format should be respected :

"line number" + " :" + "user string".

This is an example of output you could have : :

```
Exo 5
Please write the word that should be repeated 43 times
(user enters 'toto' then enter)
1 : toto
2 : toto
3 : toto
(...)
41 : toto
42 : toto
43 : toto
Exo 6.1
(...)
```

4.6 Ex 6.1 : Double for

Use this snippet under "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 } };
```

It is a 2 dimensional array. To access the first line, third column you can use tab[0,2].

You should use this array for the entire exercise 6 !

Write a program that reads the two-dimensional array with two for loops and prints it out exactly like this :

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

Note : For the first and last line with '- - - - - - ' you can put these lines of code outside the two for loops. The same applies to all of exercise 6

4.7 Ex 6.2 : Double dose of double "for"

Use the same array and two for loops but print it out exactly like this :

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

Hint : It is almost like the exercise 6.1, just two things to change.

4.8 Ex 6.3 : The revenge of double "for"

Write a program with two for loops that should print exactly this in the console :

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

If you have finished the 6th exercise you can add the following two lines :

```
Console.WriteLine("hit enter to continue");
Console.ReadLine();
```

This should be the ouput of the 6th exercise.

```
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 Ex 7 : The return of the revenge of double "for"

Under "Console.WriteLine("Exo 7")" write a program that uses two nested loops in order to compute the sum of 1 to 10.

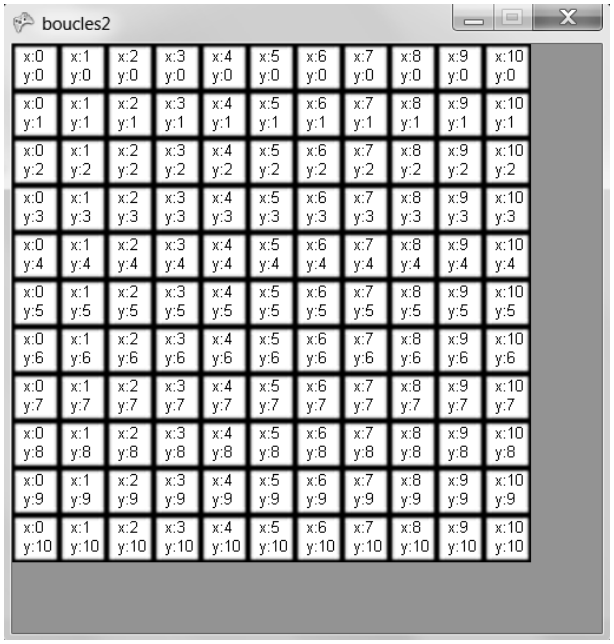
It should look **exactly** like this :

```
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 2D Loops

Sometimes your input is a one-dimensional array but you have to use it like a two-dimensional array for your calculations.

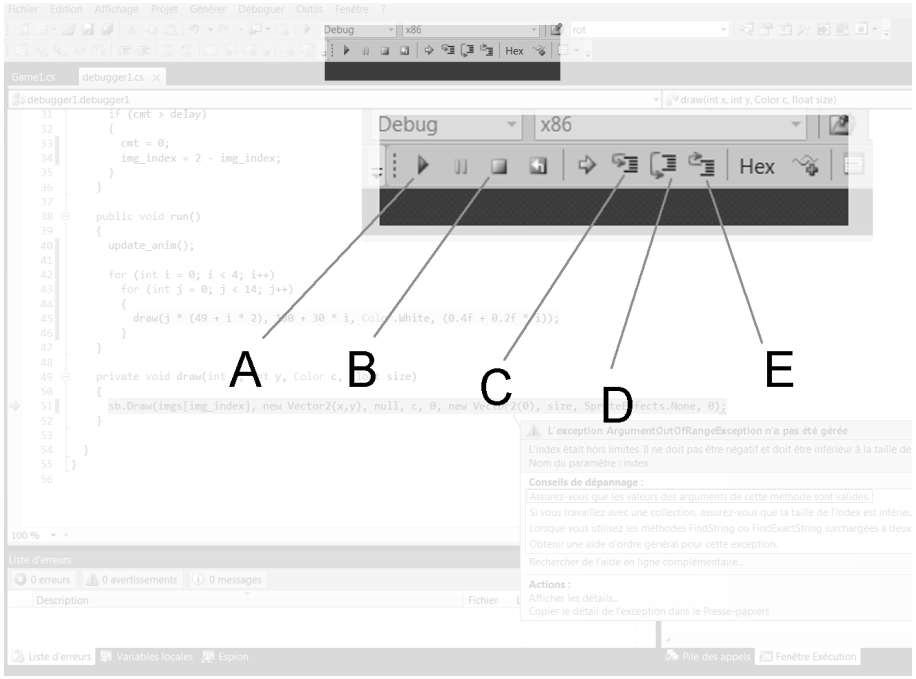
Open the "Boucles2.sln" soution. Try displaying the following image only by modifying the parameters of draw (line 36).



Hint : Use the modulo "%" and the division "/" operators.

6 Bug hunt

In big projects, you will probably spend a lot of time debugging your work. It is important to know how to work with a debugger. Open the "debugger1.sln" solution and start the execution of the program. The program will crash and you will notice you have access to new options in your menu.



6.1 A : continue

This option will try to continue the program execution. If you comment the yellow line and use continue, the program will continue its execution.

6.2 B : stop the debugger

This option will abort the execution of the debugger.

6.3 C D et E : pas à pas

To understand these options you will need to use what is called a "breakpoint".

Stop the debugger with the appropriate button.

Now click on the empty space next to the line numbers left of the line "updateAnim()".

A red dot appears : a breakpoint. When the execution flow of your program will get to this point it will pause. Launch the program with F5 and you'll see the program stop on the selected line. You can now advance step by step with (C) without entering function calls, or with (D) if you want to step into

function calls. You can use (E) to execute until the end of the current function. Please play with these three a little to understand them well.

You can also use breakpoints to be sure that some part of the code is/is not executed.

Another technique used in debugging is to isolate the error. You can prevent the execution of some instructions, or force values to test your program more thoroughly.

6.4 Debugging exercise

Try to find the bug in the debugger.cs file and correct it.

Hint : There is only one small change to make. If you succeed you should see dancing trolls.