TP C#1

1 Submission

1.1 Archive

You must submit a zip file with the following architecture :

```
– rendu-tp-firstname.lastname.zip
    |– rendu-tp-firstname.lastname/
        |– AUTHORS
        |– README
        |– Indentation/
            |– Indentation.sln
            |– Indentation/
                |– Everything except bin/ and obj/
        |– PlayWithTypes/
            |– PlayWithTypes.sln
            |– PlayWithTypes/
                |– Everything except bin/ and obj/
        |– Polish/
            |– Polish.sln
            |– Polish/
                |– Everything except bin/ and obj/
        |– PrintHouse/
            |– PrintHouse.sln
            |– PrintHouse/
                |– Everything except bin/ and obj/
```

- You should obviously replace *login* with your login (which is *firstname.lastname*).
- **The code must compile** (pressing F5 must display a console rather than compilation errors).

1.2 AUTHORS

This file should contain the following : a star (*), a space, your login and a newline.
Here is an example (where $ represents the newline) :

```
* firstname.lastname$
```

Please note that the filename is AUTHORS with NO extension.

2 Introduction

Let's start with (almost) serious things. C# is now your official weapon. Remember that you will (probably) code your project using this language. Therefore, It is mandatory for you to understand the basics that will be taught to you here.

During this practical, we will see basic notions, that should not be different from what you already know, but which will show you more about the C# syntax.

Most of the methods that you will implement here are basics. They will show you the language basis, with no need to implement complicated algorithms, and you will be able to discover how to handle the language. If any of these methods troubles you, don't hesitate to ask questions, since you will see them quite often during your years at EPITA.

During this TP, you will see the following notions:

- Arch Linux basics
- Imperative mode
- Variables
- Conditions

3 Basic knowledge

3.1 Arch Linux

You surely already have heard about the famous OS (Operating System) Linux. It would take too much time to teach you what exactly Arch Linux is so just remember it is like a "descendant" of Linux. Arch Linux is actually a distribution of Linux like Debian, Ubuntu, Fedora, Redhat...

The Linux philosophy is to use the terminal to move inside your system and be faster than clicking on icons with your mouse.

Here are some basic commands that are very useful to begin with Arch Linux:

- cd (Change Directory) - change between directories
- ls (List) - list the files in the current directory
- mkdir <dirname> (Make Directory) - create a directory
- rm <filename> (Remove) - remove a directory
- touch <filename> - create a file
- man <command> - THE programmer's manual: useful when you don't what a command is doing

3.2 Imperative mode

C# is an *imperative* language, unlike our good friend Caml. But, what are the differences between functional and imperative languages?

Let's look at Wikipedia's definition of imperative programming:

« Imperative programming is a programming paradigm that describes computation in terms of statements that change a program state. Imperative programs define sequences of commands for the computer to perform. »

Among the main differences between Caml and C#, we find the "replacement" of identifiers by variables, and the appearance of loops (which will be seen in another session).
You should remember that imperative programming can be described as a hierarchical sequence of instructions: it just seems like the “natural” way to program.

3.3 Variables

In C#, unlike in Caml, you have to tell your program the type of your variables. Forget the "let", now you have to use "int", "string" or "float". You can find below a array describing the most common types and their limits.

Some types in C#	Description
byte	Integer from 0 to 255
short	Integer from -32768 to 32767
int	Integer from -2147483648 to 2147483647
long	Integer from -9223372036854775808 to 9223372036854775807
float	Simple precision number from -3,402823e38 to 3,402823e38
double	Double precision number from -1,79769313486232e308 to 1,79769313486232e308
char	Character
string	A sequence of zero or more characters
bool	Has only two states : true and false

And if you want to use it in your code, you have to declare it like that :

```
1 int a; //  This is an int
2 float b = 42.42F; // This is a float
3 b = 0.3F // b is now equal to 0.3
4 string s = "Hello World!";
```

You do not need to assign something to your variable when you declare it. This can be useful sometimes but remember to do it before you use them or you will end up with some strange results! [1]

[1] You will see in the following TPs.

3.4 Conditions

As a programmer, you will have to check if a variable is equal, greater or inferior to an other one. To do so, you'll use conditions and operators. You have to use the famous "if else" structure and the following operators:

- a == b - is a equal to b
- a != b - is a different to b
- a < b - is a inferior to b
- a <= b - is a inferior or equal to b
- a > b - is a superior to b
- a >= b - is a superior or equal to b

Here is an example:

```
1  int a = 1;
2  if (a == 1)
3  {
4     Console.WriteLine("a is equal to 1");
5     /* ... */
6  }
7  else
8  {
9    Console.WriteLine("a is not equal to 1");
10   /* ... */
11 }
```

If you want to have the result of multiple conditions in only one expression, you can use the following operators:

- a && b - a AND b
- a || b - a OR b
- !a - NO a

Another way to do a lot of tests is to use a switch:

```
int a = 1;
switch (a)
{
    case 1:          //Check if a equals 1
        /* ... */
        break;       //DO NOT FORGET the break,
                     //otherwise it will do all the conditions
    case 2:
        /* ... */
        break;
    case 3:
        /* ... */
        break;
}
```

4 Exercises

As for the TP 0, you will use *MonoDevelop* to do the following exercises.

For each exercise, you will create a project.

To create a new project, click on "File" on the top left corner of the *MonoDevelop* window and follow these instructions:

- New
- Solution
- .NET on the left menu
- Console Project (make sure to select C# as language)
- Give a name to your project (it should follow the architecture)
- Click next

Press F5 to compile and run your project.

5 Exercise 1: Indentation and comments

Create a new project and call it "Indentation".

In this exercise, we will learn indentation and comments. Those principles will be unavoidable in all your future programming projects. In the beautiful world of imperative mode your methods should be clean!
These rules will be explained by your ACDC and you MUST respect them. To see if you have understood what is indentation, here is a piece of code that's not allowed anymore:

```
1      public int MySuperMethodOfTheDeath(int fourtyTwo)
2      {
3  if (fourtyTwo > 0)
4  {
5                                    ACDC > ASM
6          Console.WriteLine("Print something");
7  return 42;
8                  }
9      else
10 {
11 return fourtyTwo;
12 }
```

Your mission is to give to this code the beauty it deserves.

There are two kinds of commentaries in C# (like in many other programming languages).

```
1  // comment a line
2  /*
3  comment
4  a
5  block of lines
6  */
```

As you probably saw, the code above doesn't compile. Comment the line that makes no sense and make the program compile.

Of course, in real life, commentaries are there to add an explanation to your code and not to comment useless lines.

Note: In all C# TPs, we will use the MSDN[2] coding style. This coding style is used is the language standard library is used in many big C# projects. In a nutshell, classes, methods, enums, structs are named with the format 'MyNamedThing' and variables, arguments, constants, attributes are named with the format 'myNamedThing'. This name convention is called *CamelCase* (I let you guess why "Camel"...). For your general culture, in other languages like the C, we rather will use the *snake_ case*: my_function for functions and my_variable for variables.

[2]The Microsoft Developer Network (MSDN) is the section which is in charge of relation with developers.

6 Exercise 2: Play with types

Create a new project and call it "PlayWithTypes". After this exercice you will be a master at manipulating types and what you can do and what you cannot. We will ask you to code severals basics methods in this exercise.

6.1 We need fact

You have to code the famous fact method, it takes an *int* and returns the factorial of this number.

```
1 public static int Fact(int n);
2 // Fact(0) == 1
3 // Fact(5) == 120
4 // Fact(10) == 3628800
```

6.2 Pow-er rangers incoming!

Easy one, you have to code the pow method, it takes two *int* and returns the first number (x) raised to the power of the second (y).

```
1 public static int Pow(int x, int y);
2 // Pow(1, 2) == 1
3 // Pow(2, 2) == 4
4 // Pow(2, 10) == 1024
```

6.3 Hello ...!

Because it's always nice to say hi, you have to do it. Your method takes a string name and prints "Hello name!".

```
1 public static void HelloName(string name);
2 // HelloName("Edgar") -> "Hello Edgar!"
3 // HelloName("My beloved ACDC") -> "Hello My beloved ACDC!"
4 // HelloName("Mister Bond") -> "Hello Mister Bond!"
```

6.4 Even or odd

The title says it, your method must print a message saying if the number given as parameter is even or odd.

```
public static void EvenOdd(int x);
// EvenOdd(0) -> "0 is even"
// EvenOdd(53) -> "53 is odd"
// EvenOdd(1234567890) -> "1234567890 is even"
```

6.5 Different angles

In this exercise, you have to code two methods. The first one takes a angle in degrees x and return its equivalent in radian whereas the second one does the opposite.

```
public static float DegToRad(float x);
// DegToRad(1) == 0,0174533
// DegToRad(53) == 0,925025

public static float RadToDeg(float x);
// RadToDeg(1F) == 57,2958F
// RadToDeg(5.5F) == 286,479F
```

7 Exercise 3: Polish notation

Create a new project and call it "Polish".

In this exercise we will learn what is the "polish" or "prefix" notation of an arithmetic expression.

The polish expression for adding the numbers 1 and 2 is written "+ 1 2" rather than the classical ("infix" notation) "1 + 2". Another example with parentheses could be "(5 - 6) * 7" in classical notation is equal to "* (- 5 6) 7"43.

Your goal is to code a mini polish calculator.

The first method you need to code is *eval*. This methods takes the symbol of the operation and the 2 values in arguments, then returns the result.

```
1 public static int Eval(string op, int val1, int val2);
2 // Eval("+", 4, 7) == 11
3 // Eval("/", 10, 2) == 5
4 // Eval("*", 10, 2) == 20
```

Now that we have our eval method we can code the *main* method. The method will ask for the symbol of the operation (+, -, /, *, %) and 2 values to the user and then call the *eval* method with them.

Tips: You can use *string Console.ReadLine()* to read the user's input and *int Convert.ToInt32(string value)* to convert a *string* into an *int*.

```
1 public static void Main(string[] args)
2 {
3         // FIXME
4 }
```

You should have the following output in the console when pressing F5:

```
Enter an operator:
/
Enter the first value:
28
Enter the second value:
7

/ 28 7 = 4
```

8 Exercise 4: Print house

Create a new project and call it "PrintHouse".

Do you really know how to print in the console ? That's what we are going to check. In this exercice, you will be asked to print a house. To do so you need four methods.

8.1 The roof

Print the roof of your house, using the following prototype:

```
1 public static void roof();
```

It's your roof so you can do whatever you want but it has to looks like a real roof. An example can be :

```
1   _____________
2  /  |  |    |  |  \
3 ---------------
```

8.2 The floor

Print the floor of your house, use the following prototype:

```
1 public static void Floor();
```

Same here, do what you want but be realistic. You can create multiple floors and print them using the method you want.

```
1 |   |0|    |0|   |
2 |                |
```

8.3 The ground

Print the ground of your house, use the following prototype:

```
1 public static void Ground();
```

Same here, do what you want but be realistic.

```
1 |        _        |
2 |_______|-|_______|
```

8.4 Print everything

Okay, here is the final method! You have to code the method PrintHouse using the previous methods. Here is the prototype:

```
1 public static void PrintHouse();
```

Final result:

```
1   _____________
2  /  |  |    |  |  \
3 ---------------
4 |   |0|    |0|   |
5 |               |
6 |       _       |
7 |_____|-|_____|
```

8.5 Bonii

If you have finished all your TP, you can add some bonii! They can be anything but here is one example: different patterns for the floor (but you can find other things to do). Be creative!

The Code is the Law.