TP C#2:

1 Hand in instructions

At the end of this practical work, you will have to hand in an archive file following these instructions:

```
rendu-tpcs2-login_x.zip
|-- rendu-tpcs2-login_x/
    |-- AUTHORS
    |-- tpcs2.sln
    |-- tpcs2
        |-- Everything except bin/ et obj/
```

Of course you have to replace "`login_x`" with your own login.

Don't forget to check the following items before handing in:

- The `AUTHORS` must follow the usual format (a *, a space, your login and a newline character).
- No `bin` or `obj` directories in the project.
- **The code must compile!**

2 Introduction

2.1 Objectives

During this practical work you will study the following topics:

- The use of the console in C#
- The Windows terminal
- Powershell
- The console on Unix
- The Main function

3 Course

3.1 The console

As a good Windows user, you like to click on buttons with your mouse. But you may not know that another world exists – faster, darker, a world that will make any average person think that you're a hacker: the console.
Indeed, in the old days[1], computer scientists had at their disposal only a command line interface on a screen displaying only 80 columns.
Despite the massive use of graphical interfaces (GUI = Graphical User Interface) nowadays, the console and its text commands still remain thanks to its efficiency.

3.2 The Windows terminal

Every Windows version has a console inherited from MS-DOS that you can open by running **cmd.exe** (with Win+R for instance): the Windows terminal.
In this terminal you can execute a whole lot of commands and group them in files to automate actions easily.[2]
Here are the main commands:

dir
: Lists the files in the current directory.

cd
: Changes the current directory.

copy
: Copies a file to another place.

mov
: Moves a file or directory.

print
: Prints a file.

You an also run an executable just by typing its name (prefixed by the path to it if different from the current one). This interface is very limited and outdated and is only kept for compatibility reasons.

3.3 Powershell

As its name says –a shell is a command interpreter, it is a new version of the Windows terminal introduced in Windows 7.
Its functionalities are closer to their Unix equivalents with similar command names and the appearance of advanced functions like stream redirection.[3]
The previously explained commands become the following:

[1] Earlier... 1970!
[2] Yes, a .bat file is not only used to destroy your computer.
[3] http://www.tldp.org/LDP/abs/html/io-redirection.html

ls
Lists the files in the current directory.

cd
Changes the current directory.

cp
Copies a file to another place.

mv
Moves a file or directory.

cat
Prints a file.

We also notice the appearance of **man** that allows the user to get all the necessary information about a command just by typing man before.

3.4 The console on Unix

From next year (or right now if you want!) you'll be using Unix compliant operating systems, in which the command line interface is still very present and has been developed extensively in order to offer unlimited use.
One of the reasons why the vast majority of the servers run on this kind of OS is that when you connect to them you only have access to a TTY.[4]
We will not speak in detail about all the wonders you can find in these environments but if you're interested don't hesitate to search on the internet or ask your ACDC.

3.5 The use in C#

The management of the console in C# is done through... the Console class!
Its reference is available on the C# bible: MSDN.
You can access it here: http://msdn.microsoft.com/en-us/library/system.console
Your first work in this practical will be to search (not necessarily in the above page) at least the following functions to understand how to use them:

1. Console.Write
2. Console.WriteLine
3. Console.Read
4. Console.ReadLine
5. Convert.ToInt32
6. Convert.ToFloat

3.6 The Main function

When you will compile and run your program, the first and only function that will be called is `Main()`. This function is the entry point of your program, so if you want to call your functions, you have to do it here.

[4] http://en.wikipedia.org/wiki/Teleprinter

4 Exercises

4.1 Before you start

While doing these exercises you will often be asked to display text on the console, without any other indication you have to add a new line at the end of every exercise.
You will also notice that if you run your program in Visual Studio, it will spawn and disapear before you can do anything. To fix this problem, you can either add a call to `Console.Read()` at the end of your `Main` in order to wait for some user input, or simply launch your program from a console.

It is now time to start the real work, that is creating a new console project in Visual Studio by selecting File/New/Project/Console Application with "tpcs02" as a name and going on with the following exercises!

4.2 Exercise 0: Hello World

Write the `HelloWorld` function that displays the text "Hello World!" in the console.
Prototype:

```
public static void HelloWorld()
```

Example:

```
> test_helloworld.exe
Hello World!
```

4.3 Exercise 1: Hey ! Echo... cho... o...

Write the `Echo` function that reads a line typed by the user and prints it afterwards.
Prototype:

```
static void Echo()
```

Example:

```
> test_echo.exe
test
test
```

4.4 Exercise 2: Esrever

Write the `Reverse` function that reads a line and prints it reversed afterwards.
This function must be imperatively recursive.[5]
PROTIP: let s be a string, `s[i]` is the character at index i (beware, the indexes start at 0) and `s.Length` is the length of s.
The Reverse function may call another function to do the recursion.
Prototype:

```
static void Reverse()
```

Example:

```
> test_reverse.exe
test
tset
```

[5] And not recursively imperative.

4.5 Exercise 3: Triforce

Write the `Triforce` function that prints a triforce in yellow on a blue background and changes the title to "Triforce".
You must also call `Console.ResetColor()` at the end to reset the console to its normal state.
PROTIP: these changes are not done through function calls but by modifying the `Console` properties (look for "Console Class" on MSDN). Also search for `ConsoleColor`.

Prototype:

```
static void Triforce()
```

Example:

```
> test_triforce.exe
   /\
  /__\
 /\  /\
/__\/__\
```

Bonus: Use only one `Write`, because one line is better.

4.6 Exercise 4: MCQte

Write the `QCM` function that asks the user a question and prints:
- "Good job bro!" if the answer is right.
- "You lose... The answer was X." if the answer is wrong, replacing X by the number of the right answer.
- "So counting is too hard, n00b..." if the input is invalid.

The `ansX` options only contain the text of the answer, it is your job to print the "X) " at the beginning of each one.
The `answer` parameter contains the number of the right answer.
Do not handle the cases in which the user doesn't type a number.
PROTIP: to concatenate two strings you can use the operator `+`.

Prototype:

```
static void QCM(string question,
                string ans1,
                string ans2,
                string ans3,
                string ans4,
                int answer)
```

Example:

```
> test_QCM.exe
What's the difference between a pigeon ?
1) The legs, especially the left one
2) Yes
3) Obiwan Kenobi
4) The D answer
2
You lose... The answer was 1.
```

Bonus: handle incorrect input (search how to use `try ... catch`).

4.7 Exercise 5: ___ _____ ._. ... .

Write the `Morse` function that produces the sound corresponding to the message given in input.
This message will be constituted of the '_', '.' and ' ' characters only.
A '_' correponds to a 450ms beep, a '.' to a 150ms beep and ' ' to a 450ms pause.
The sounds must have a frequency of 900Hz.
This function must, as usual, be recursive.
PROTIP: you can use `Console.Beep(f, n)` to produce a sound of frequency f Hz during n milliseconds and `System.Threading.Thread.Sleep(n)` to pause the program for n milliseconds.

Prototype:

```
static void Morse()
```

Example:

```
> test_morse.exe
.___ .____. ._ .. __ . __ . ... ._ _._. _.. _._.
bip biiip biiip... (nothing is displayed)
```

4.8 Bonus 1: Draw me a pony

Write the `DrawPony` that prints the best ASCII art pony ever with colors and anything else you want.

Prototype:

```
static void DrawPony()
```

Bonusception: animate your pony!

4.9 Ultimate bonus: sl

Write the `Sl` function that reproduces the behavious of the **sl** Unix command.
PROTIP: Google is your friend.
Bonusception: handle command line arguments.
(look at `string[] args`,the parameter of `Main`)

It's dangerous to code alone !