# TP C#6
# My Explorer

## Submission

### Archive

You must submit a zip file with the following architecture :

```
rendu-tp-firstname.lastname.zip
    |-- rendu-tp-firstname.lastname/
        |-- AUTHORS
        |-- README
        |-- Crypto/
            |-- Crypto.sln
            |-- Crypto/
                |-- everything except bin/ and obj/
        |-- MyExplorer/
            |-- MyExplorer.sln
            |-- MyExplorer/
                |-- everything except bin/ and obj/
```

- You should obviously replace *firstname.lastname* with your name.
- The code must compile (There should not be any warning).

### AUTHORS

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

```
* firstname.lastname
```

Please note that the filename is AUTHORS with NO extension.

### README

This file should contain information about your TP: what did you manage to do, where did you have a hard time, **what bonii have you done**...
Please note that the filename is README with NO extension.

# 1 Introduction

## 1.1 Objectives

During this TP, you will learn new notions such as:

- Manipulating files
- The "using" statement
- Moving through the filesystem

# 2 Course

## 2.1 Manipulating files in C#

Up until today, you have been creating programs that only ran once (twice when your ACDC corrected them), and then disappeared in the emptiness of the void. Poor programs, they never got the chance to experience the joy of manipulating all those wonderful files on your hard drive (or your RAM, in the case of the school's computers).
Well this TP is here to change that, soon you will be able to communicate with the rest of the world[1].

#### The easy part: The `File` Class

The `File` Class[2] is a class that allows you to to manage files (surprisingly). Its main purpose is to manage the files themselves, not really their contents. Here is a small list of the most used methods in this class:

```
1 public static void Delete(string path);
2 public static FileStream Create(string path);
3 public static void Copy(string sourceFileName, string destFileName);
4 public static FileStream Open(string path, FileMode mode);
5 public static void Move(string sourceFileName, string destFileName);
```

You might have noticed that the second method (`File.Create`) returns a strange object: a `FileStream`. Basically a `FileStream` is a way for a C# program to represent a file. In a program you can not just take a file, load it in your memory as part of your program, and then do what you want with it. Files have permissions that limits your ability to read them or write to them, depending on who you are. But, most importantly, Files have a size: imagine if you want to read a 1 Gb file. You would have to load 1 Gb of data from your disk to your RAM. This is slow, and if every program did that, watching high quality movies would require a **lot** of RAM.
So instead of loading the complete file, we use a `FileStream` object in C#, which is an object that will interact with a file the way you want it, and do all the heavy lifting for you. A lot of file managing methods take a `FileStream` as argument, to know which file you want them to interact with.
Moving a file around is fun, but won't get you very far. You want to read files, and to write or rewrite to them as well. To achieve that there are multiple ways:

[1] Limited to your local machine, you'll have to wait a bit before going online!
[2] https://msdn.microsoft.com/en-us/library/system.io.file(v=vs.110).aspx

- You use the `File.ReadAllLines` or `File.ReadAllText` methods, store the contents into a variable, do whatever want you want with it, and then write back to the file using `File.WriteAllLines` or `File.WriteAllText` methods.
- You use a `StreamReader` to read the file, and a `StreamWriter` to write in the file.

While seeming simpler, the first method is actually not very useful. You are indeed storing in a variable the whole contents of a file. Whereas this can be suited for small files or configuration files, it is far less acceptable for arbitrarily-sized files. So, as you might have guessed, you will use the second method: The `StreamReader` and `StreamWriter`.

#### The fun part: `StreamReader` and `StreamWriter` classes

The `StreamReader` class[3] manages Reading in a file, of any format (it reads directly the bytes from a file), but don't worry, it is perfectly adapted for Reading in a text file as well, since a character is only a byte. In the example, there is a file named `mytreasuredfile.gold` in the same folder as the program executable. This file contains only the following line:

```
This is either madness... or brilliance...
```

And here is a sample of code that could read it:

```
// We create a new StreamReader object named sr, which will read the file
// named mytreasuredfile.gold
StreamReader sr = new StreamReader("mytreasuredfile.gold");
// We can read a character from the file using sr.Read()
// since the file is read sequentially (from the beginning to the end),
// here we read the first character
char character = (char) sr.Read();
// -> character == 'T'
// We read the whole line until the end.
// Since we read the first character before, here it is missing
string line = sr.ReadLine();
// -> line == "his is either madness... or brilliance..."
// sr.EndOfStream is true if we read everything, false otherwise
// Here it is true since we read one line and the file contained only one
if (sr.EndOfStream)
  Console.WriteLine("The file has been read completely");
else
  Console.WriteLine("There is still something to read");
// We have to close the StreamReader
sr.Close();
```

Don't hesitate to ask your ACDCs if you have any question with the code above, as it is very important for you to understand it. You should also have a look at the following methods as they might prove

[3] https://msdn.microsoft.com/en-us/library/system.io.streamreader(v=vs.110).aspx

useful for the TP: `Peek()`[4] and `Read(char[], int, int)`[5].

If you understood how the `StreamReader` works (and if you did not, ask your ACDCs about what you did not understand), then the `StreamWriter` class[6] should be very easy to understand, this class works the same way as the `StreamReader`.
Here is a quick example:

```
// We create a new StreamWriter object named sw, which will write to the
// file named stuff
StreamWriter sw = new StreamWriter("stuff");
// We can Write a character to the file using sw.Write()
// This works the exact same way as the Console.Write() method
sw.Write('a');
sw.Write(14);
sw.Write(13.37);
// We can also use sw.WriteLine() to write a whole line.
// It also works like Console.WriteLine();
sw.WriteLine("The code is the law");
sw.WriteLine("The answer is {0}", 42);
// We have to close the StreamWriter
sw.Close();
```

If we have a look at the stuff file now:

```
a1413.37The code is the law
The answer is 42
```

Again, you have to understand this example before continuing reading the TP.

### 2.2 Use me !

You might have noticed that, in the previous examples, you had to close the stream you were using, but why ? That is because what you do is not immediately done by the computer: behind the scene, when you write something, it is not written immediately to the file but instead to a buffer, which will be flushed at some point (you can't know when), so if your programs ends without closing properly the file you will end up with the file not being completely written. This is were the `Close()` method is useful.
Still having to write `Close()` each time can be annoying, and you can quickly forget to do it. For that reason you should always use the `using`[7] statement. This statement simply tells the compiler to automatically and properly dispose of the variable at the end of the scope. Here is how you can use it:

```
using (StreamWriter sw = new StreamWriter("file"))
{
```

[4] https://msdn.microsoft.com/en-us/library/system.io.streamreader.peek(v=vs.110).aspx
[5] https://msdn.microsoft.com/en-us/library/9kstw824(v=vs.110).aspx
[6] https://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.110).aspx
[7] https://msdn.microsoft.com/en-us/library/yh598w02.aspx

```
  // sw can be used to do whatever we want with it
  sw.WriteLine("Gentlemen!");
  //...
  // sw.Close() will be automatically called before
  // leaving the scope of the using
}
// sw is not available anymore
```

### 2.3 Taking a stroll through the Filesystem

Being able to create, read and write file is definitely cool, but it is still not enough. You're currently limited to your current directory: the one you launched your program from. What if we want to move around, or to see what is in a directory ? To do so you can use the following methods, all defined in the `Directory` class[8]:

```
public static string GetCurrentDirectory();
public static string[] GetDirectories(string path);
public static string[] GetFiles(string path);
public static string[] GetFileSystemEntries(string path);
public static void SetCurrentDirectory(string path);
```

Their usage will not be detailed here as they are all pretty self-explanatory. If you still have any doubt, you can always check the MSDN for those methods or ask your assistants.

Please note that if you wish to use any of the functions that were talked about in the course section of this TP, you have to put the following line at the beginning of your `Program.cs` file:

```
using System.IO;
```

Now that you've read through all that, and understood it all (if you did not, start over or ask questions), you should be ready for the exercises. Good luck!

## 3 Exercise 1: Crypto

Cryptography is a means to make a message impossible to understand to anyone not possessing the necessary information to decode it. As such, this process goes both ways: an encrypted message can be decrypted, as opposed to hashes like MD5 or SHA. The most basic methods of cryptography are **rotn** and the **Vigenère** cypher. Nothing really new: those have already been done in Caml, but you will have to implement them again in C#.
The following functions have to be coded in the same file.

[8] https://msdn.microsoft.com/en-us/library/system.io.directory(v=vs.110).aspx

### 3.1 Rotn

**Rotn** is a method that shifts every character of a text by n positions. The function we ask you to implement will simply shift letters and digits of a string passed as argument.

```
static string Rotn(string msg, int n);
//Rotn("abCz18", 3) -> "deFc41"
//Rotn("deF ;c41", -3) -> "abC ;z18"
//Rotn("ACaC1", 101) -> "XZaZ2"
```

Tip: Use the modulo operator ('%') and the `ASCII` table

### 3.2 Vigenère

The **Vigenère** cypher is another method of encryption which will change each letter of a text into another one thanks to a given key. The particularity of it is that a letter will not always be output to the same letter, depending on its location in the text. This method needs the "Vigenère table", but to save you the Wikipedia search, we give you a reminder of the rules you must follow to encrypt and decrypt a text. You must respect the following rules:

- The key and the message are of variable length and can be composed of any character of the ASCII table.
- **In the message**, only letters are cyphered. An uppercase letter gives an uppercase letter, a lowercase one gives a lowercase one. The other ones are skipped and the position in the key stays unchanged.
- **In the key**, only letters will serve in cyphering/decyphering. Lowercase letters must be considered as uppercase ones.
- `Cyphered[i] = (Text[i] + Key[j]) modulo 26`
- `Text[i] = (Cyphered[i] - Key[j]) modulo 26`

```
static string VigenereEncode(string msg, string key);
//VigenereEncode("AB xza ,;48ACk", "Un petit post-scriptum pour vouS")
//  -> "UO mdt ,;48IVz"

static string VigenereDecode(string msg, string key);
//VigenereDecode("AB xza ,;48ACk", "Quatoorze ;z")
//  -> "KH xgm ,;48ML1"
```

## 4 Exercise 2: MyExplorer

This exercise will put in practice the course at the start of the TP. You will have to read files, and modify them. The goal is to have a really basic interactive program which will be able to execute very simple commands such as: `ls`, `cd` or `cat`. You will also see how to parse a very simple file format.

Note: We will only test your functions with files in the correct format, except for threshold 4. You can as such deduce that the text given in every file will match exactly what is given in the instruction for the threshold in question. Moreover, for the whole exercise, no key or value can contain the character ':' or ','. Sample files will be given to test your exercise on the intranet `acdc.epita.it`. For the whole exercise you are not allowed to use the `RegExp` class, you can however (and are strongly advised to) use the `String`[9] class.

For the whole exercise, if, for any reason, you can't open a file (for example it does not exist), you have to write the following on the console's standard error output: `"Could not open file: "` followed by the name of the file.

### 4.1 Threshold -1 : MiniCat

This threshold is meant as an introduction to `MyExplorer`. The function takes the name of a file as argument. The goal of this function is to open it, read it and write its content on the console's standard output. You will also provide another function the goal of which is to write the content of the file to the file given as second argument. If it does not exist you should create it, otherwise you should append the content at the end of the file.

```
1 static void MiniCat(string InputFile);
2
3 static void MiniCat(string InputFile, string OutputFile);
```

### 4.2 Threshold 0: GetValue

Now that you are familiar with reading inside a file, let's start the real exercise. In this exercise, you will have to get (and potentially modify) a value which corresponds to a key. For this threshold you will have to provide the `GetValue` function, which takes two arguments: the name of the file and the name of the key. Each line of this file will be made the same way: a key, followed by ':', followed by a value, followed by a line return. This file does not not contain any empty line, there is not any space between each element.

```
1 static string GetValue(string InputFile, string key);
```

### 4.3 Threshold 1.0: MiniLs

To validate this threshold, you have provide the `MiniLs` function. The goal of this function is to write on the console every file/folder of the current folder.

```
1 static void MiniLs();
```

[9] https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx

### 4.4 Threshold 1.5: cd

To validate this threshold, you have to provide the `MiniCd`. This function takes the name of a folder as argument and will change the current directory to this folder. If the argument is invalid, the function will write an error message on the console's standard error output.

```
1 static void MiniCd(string FolderName);
```

### 4.5 Threshold 1.8: Main function

To validate this threshold, you will have to make the program usable. Your `Main` function will read the user's input in an infinite loop, like a real terminal (the big black box for the lost ones...). The user will be able to write `cat`, `ls`, `cd` or `get` followed by arguments separated by spaces. The arguments will not contain contain any space or quotes (simple or double). For this threshold, you can assume that that there won't be any error in the argument number for any of the commands. The `cat` command calls `MiniCat`, the `ls` command calls `MiniLs`, `cd` calls `MiniCd` and `get` calls `GetValue`.

### 4.6 Threshold 2: Let's add some spaces

To validate this threshold, you have to handle files containing spaces before and after each element in the `GetValue`. Have a look at the files given on the intranet to see exactly where the spaces can appear. The previous threshold must still be functional after this one.

### 4.7 Threshold 3: Multiple values per key

To validate this threshold, you have to create a second version of the `GetValue` function, which is the position of the asked value. The files given will always have one key per line, but there may be more than one value per key, separated by commas. The first value has position 0.
Your `Main` should also handle `get` with 3 arguments now.

```
1 static string GetValue(string InputFile, string key, int position);
```

### 4.8 Threshold 4: Error Management

To validate this threshold, you must handle errors in you `GetValue` functions when there is an obvious error in the file. An error must be returned when:

- The key is not present
- The key has no value
- You are trying to access a value that does not exist (for example the key has 3 value and you try to access the value at index 3)

### 4.9 Threshold 5: Change is now

For the last threshold you must be able to change a value in a file. For example: a file contains the line "ACDC: Acdc, are, Bad, guys". Calling `ChangeValue(InputFile, "ACDC", "Bad", "Good")` will change the line to "ACDC: Acdc, are, Good, guys". As you can see, the number of spaces should be preserved, only `OldValue` is changed. `OldValue` will not appear more than once, but may not appear in the line. If that's the case, do nothing.

```
static void ChangeValue(string InputFile, string key,
  string OldValue, string NewValue);
//OldValue and NewValue cannot start or end with spaces.
```

Your `Main` should now handle the `change` command which will call this function.

## 5 Bonii

Here is a list of bonii advised for this TP. It is advised to do them in this order. As the are based on the whole TP, you have to have finished the first part before starting the bonii.
Don't forget to tell us what you've done in your README.

- Handle cases of wrong number of arguments in the `Main` function.
- Do the `MiniCatCrypto` function: it will read the file given as argument, look at its first line and act accordingly. If the first word is "ROTN", it will apply the rotn function to the whole file (except for the first line) using the number given as second word. It will output to the console. If the second word is not a number, you should return an error. If the first word is "VIGENERE", you will **decypher** the file using the key given as the rest of the first line. If the first word is something else, follow the normal behaviour of `MiniCat`.

```
static void MiniCatCrypto(string InputFile);
// First line of InputFile:
//   "ROTN 4z" -> error
//   "ROTN 42 4" -> rotn(42) on the whole file but the first line
//   "VIGENERE Slam du rible"
//      -> vigenère with key "Slam du rible"
//   "ACDC are the best" -> MiniCat(InputFile);
```

- Redo this TP without the methods in the `String` class.
- Redo this TP without using the `string` type (hint: char array) (hint2: Have fun).

**The code is the law.**