Case C# 0: Discovery

Assignment

Archive

At the end of the practical, your repository must follow this architecture:

```
tp0-firstname.lastname/
|-- README
|-- .gitignore
|-- Basics/
    |-- Basics.sln
    |-- Basics/
        |-- Basics.cs
        |-- Program.cs
        |-- Everything except bin/ and obj/
```

Do not forget to check the following things before submitting your work:

- Replace `prenom.nom` by your login

- The `README` is mandatory.

- No `bin` or `obj` in the project.

- Respect the given prototype for every function.

- Remove all tests from your code.

- **The code must compile !**

README

In this file, you can write any and all comments you might have about the practical, your work, or more generally about your strengths and weaknesses. You must list and explain all the bonuses you have implemented. An empty `README` file will be considered as an invalid archive (malus).

1 Introduction

1.1 Objectives

Congratulations for completing your first half of S1! Seeing that you have come a long way in OCaml, it is now time to change our environment and focus on the marvellous coding language of C#. From now to the end of S2, you will have all your practical work in C# and no longer in OCaml (this doesn't count for the AFIT project which will be in OCaml).

As you will find out soon, C# and OCaml are both languages which share some commonalities but also differ greatly. Our objective for this practical work is to transition smoothly from OCaml to the new world of C#. If you have any worries about OCaml or still feel that you have not fully embraced all the concepts of it, fear not! This transition will help to open your eyes to what is truly there.

2 Course

2.1 What is C#?

> *C# is a general-purpose, modern and object-oriented programming language pronounced as "C Sharp". It was developed by Microsoft led by Anders Hejlsberg and his team within the .NET initiative and was approved by the European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). C# is among the languages for Common Language Infrastructure. C# is a lot similar to Java syntactically and is easy for users who have knowledge of C, C++ or Java.*

https://www.geeksforgeeks.org/csharp-programming-language/

As can be seen with the definition above, C# is deemed to be an 'object-oriented', language but what exactly is 'object-oriented' you may ask? This will be explained in the section below but for now you have to know that C# is part of the C programming language and with it you will be able to achieve many goals.

2.2 Programming Paradigms

2.2.1 Definition

Some of the most important questions you may be asking yourself are what is the difference between C# and OCaml? If they are both programming languages which allow us to code different things, then why are they different? This is where we introduce the notion of *programming paradigms*

> *Programming paradigms are a way to classify programming languages based on their features.*

https://en.wikipedia.org/wiki/Programming_paradigm

2.2.2 Declarative & Imperative

More specifically, we say that a programming language belongs to a programming paradigm if it shares that paradigm's specific features. These different types of paradigms can be split into two parts, either *imperative* or *declarative*.
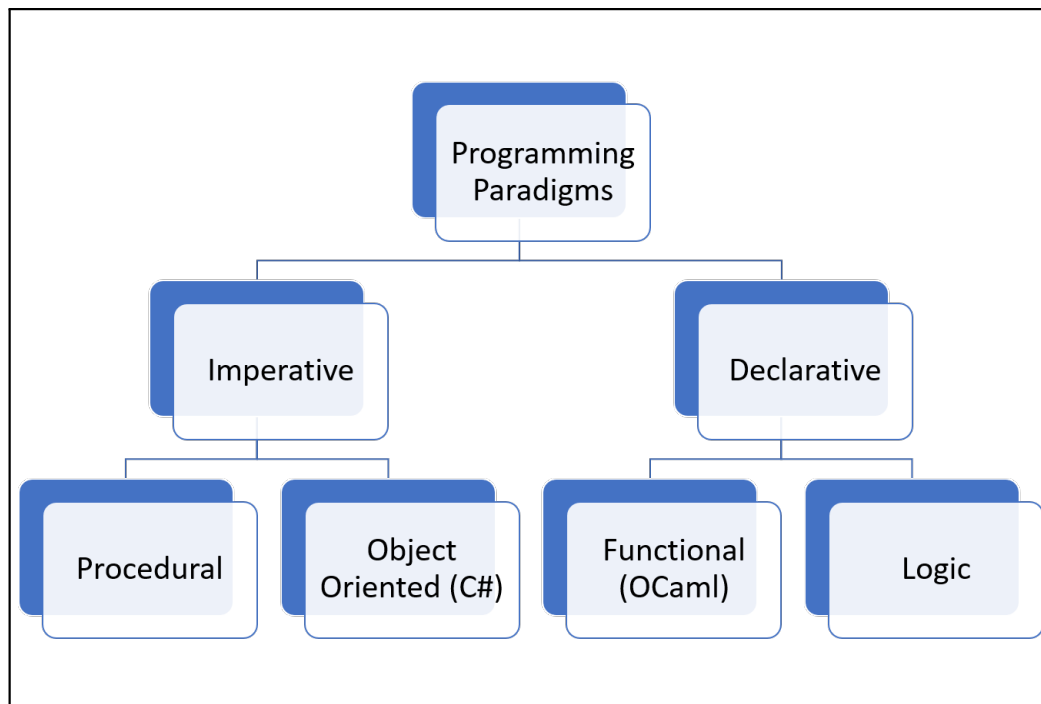
Figure 1: Diagram of programming paradigms

The main difference between the two is that an *imperative* program has a clearly-defined sequence of the user's statements to execute in order to achieve something, whereas a *declarative* program focuses on what the program should accomplish without specifying how the program should do it.

As time goes on it will become more apparent what programming paradigms are but for now you just have to know that OCaml is a *functional* programming language and that C# is a *object-orientated* language. Now we can see how C# and OCaml are different from each other.

2.3   Rider : a 'Visual Studio'-like environment

Just as you have used emacs to code and execute OCaml programs, you will be using a new code editor to do all your practical work in c#. This new editor is called *Rider* and it can be found on all the school's computers. *Rider* is an IDE (Integrated Development Environment) which contains a lot of useful features and software such as a compiler and debugger.

2.3.1   Solutions

A *solution* in C# vocabulary corresponds to what we would call a project, but *solution* can actually contain several *projects*. Typically, you will have one solution per practical, possibly organized in a few projects. You must always pay attention to the architecture required by the subject!

2.3.2   Compiling & Running

If you click on `Run`⟩`Run 'Default'`, your program will be compiled and executed. The green play button visible in the toolbar is a shortcut for this action. You can also use a keyboard shortcut, which by default is `Ctrl`+`F5`.

2.4   Types and Variables

In OCaml, you had to use the `let` keyword to bind a value to an identifier. This association was definitive, and lasted to the end of the scope. In imperative programming, values are stored in *variables*, which can be affected to different values throughout their lifetime. Variables have a name and a *type*, which determine what they can contain.

To create a variable, you must first **declare** it. This step will give the variable a name and a type, which must be explicitly specified:

```
1        string str  = "hello";
```

You can **assign** a new value to your variable using the `=` operator:

```
1    str = "good bye";
2    str = "good morning";
```

You can also **initialize** a variable by declaring and assigning it simultaneously.

```
1    int foo = 42; // Declaration and initialization of "foo".
```

Here are, among the built-in types of C#, those that you will use the most:

Type	Example value
bool	true
char	'*'
int	42
long	$9.2.10^{18}$
float	42.03
string	"Forty-two"

Table 1: C# Types

> **Going further**
>
> You will also stumble upon the `uint` type. Unlike the `int` type that is signed, `uint` is unsigned so it takes as many values as `int` but only positives ones.
> You might need to know that `double` works like `float`, but takes more space in memory while being capable of holding larger and/or more precise values.

2.5   Operators

Here is a summary of C# operators and their OCaml equivalents. The syntax of subtraction, multiplication, and division follow that of addition

Operation	C#	Caml
Summing integers	+	+
Summing real numbers	+	+.
Concatenating strings	+	^
Modulo	%	mod
Logic "And"	&&	&&
Logic "Or"	||	||
Logic "Not"	!	not
Equality	==	=
Inequality	!=	<>
Strict comparison	<	<
Comparison	<=	<=

Table 2: Operators in C# and in OCaml

> **Going further**
>
> Each of the basic arithmetic operators (+-*/% ...) has an associated assignment operator.
> For instance, `a = a + b;` is equivalent to `a += b;`.

2.6 The "if" Statement

You can test a boolean condition and choose to execute a block of code only if it is true using the `if` statement:

```csharp
if (b == 0)
{
  i = 2;
  return i;
}

if (c == 4)
  return 3; // Braces are not mandatory here.

if (b < 0)
{
    sign = -1;
    b = -b;
}
else
{
    sign = -1;
}
```

2.7   The "switch" Statement

To reduce nesting of `if` and `else` statements, one can use a `switch`.

```
switch (a)
{
case '*':
    res = x * y;
    break;
case '+':
    res = x + y;
    break;
default:
    res = 0;
    break;
}
```

It might remind you of the the `match` expressions in OCaml. However, there are a few differences between the two which are mainly syntactical.

- You can only use constant values after the `case` keyword.

- The switch statement will jump to the appropriate `case` label and will execute everything from here to the `break` keyword.

- It is possible to have multiple cases with the same behavior as seen below.

```
switch (argument)
{
case "étoile":
case "star":
    VicsekStar(...);
    break;
case "croix":
case "cross":
    VicsekCross(...);
    break;
default:
    // Error message.
    break;
}
```

2.8   Functions

A function declaration in C# can be cut in four fields:

- some optional keywords controlling who will be able to see the function

- a return type, or `void` if the function returns nothing

- a name

- a list of parameters, with their names and types

A function can be called from another function by using its name followed by an argument list under parenthesis.

```csharp
/* This function returns nothing, its return type is void */
public static void MyFunc(int arg1, float arg2)
{
    // Do magic stuff
    return;
}
public static int AnotherFunc()
{
    MyFunc(0, 4.2f); // Calling MyFunc
    return 666;
}
```

2.9 The Main Function

Your program must define a `Main` function that will be the first to be called at the beginning of your program. The program also ends when the `Main` function returns. Several prototypes are valid for this function:

```csharp
public static void Main()
public static void Main(string[] args)
public static int Main()
public static int Main(string[] args)
```

You don't have to put anything in your `Main` for this practical, but you might find it useful for testing your other functions.

3   Exercises

Every function has to be written in the `Basics.cs` file from the `Basics` solution. A skeleton for the TP is available on the Moodle page.

> **Warning**
>
> For this TP, loops are strictly forbidden and will be penalized

3.1   Hello World

You must display the following text in the console, followed by a newline:
  "Hello World!"

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

> **Hint**
>
> Check the **Console.Write()** and **Console.WriteLine()** functions on MSDN.

3.2   Welcome to 221B Baker Street

Welcoming someone is nice, but getting to know their name is nicer. You now have to ask for the user's name before welcoming them to 221B Baker Street
  Example:

```
1  Hello, what's your name?
2  Mr. Lestrade
3  Welcome to 221B Baker Street, Mr. Lestrade!
```

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

> **Hint**
>
> Check the **Console.Read()** and **Console.ReadLine()** functions on MSDN.

> **Warning**
>
> When you are asked to write something in the console, you MUST respect the same text than the subject.

3.3   Computing an age

Sherlock would like to know the age of the Irregular of Baker Street.
Let's write a function to help him do that!
(Assuming they will never have a negative age)
  Example:

```
1  What's your year of birth?
2  2003
3  Looks like you're around 18!
```

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

> **Hint**
>
> Regardez la fonction **Int32.Parse()** sur MSDN.

3.4 Power

The function `Pow` returns the number `x` to the power `n`. Beware! You have to handle negative powers.

```
1  public static double Pow(double x, int n);
```

3.5 Factorial

You must write a function returning the factorial of the number `n` given as a parameter
Reminder:
n! = 1 if $n \leq 1$ else n *(n - 1)!

```
1  public static uint Factorial(uint n);
```

3.6 IsPrime

You must write the function `IsPrime` that takes an integer `n` as an input, and returns `true` if that number is prime, or `false` otherwise. 1 is not a prime number. The first prime number is 2.

```
1  public static bool IsPrime(uint n);
```

3.7 Fibonacci

You should know it by heart now ! You must write a function that takes the rank of a number in the Fibonacci[1] sequence, and returns this number.
In case you forgot (somehow), here is how the Fibonacci sequence is defined.

- `Fibonacci(0) = 0`

- `Fibonacci(1) = 1`

- `Fibonacci(2) = Fibonacci(1) + Fibonacci(0)`

- `Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)`

```
1  static uint Fibonacci(uint n);
```

---

[1]`https://fr.wikipedia.org/wiki/Suite_de_Fibonacci`

3.8   Sherlock Holmes

Finally, you will implement a *Fizz Buzz*. It's a game where you have to count from 1 to **n**, but with a couple extra rules.

When you're on a multiple of 3, you have to say "Sherlock", and if the number is a multiple of 5, you have to say "Holmes".

Finally, if the number is a multiple of 15, you have to say "Sherlock Holmes".[2]

In this exercise, you'll have to write a function that takes an integer **n** as a parameter and returns the **string** of every step of the game.

```
public static void SherlockHolmes(uint n);

SherlockHolmes(1) == "1"
SherlockHolmes(5) == "1, 2, Sherlock, 4, Holmes"
SherlockHolmes(15) == "1, 2, [...], 14, Sherlock Holmes"
```

The returned string me not contain any line break. If n is null, the returned string is supposed to be empty.

---

[2]This is a modified version of `Fizz Buzz` a traditional programming exercise

4 Bonus

4.1 Ave Caesar

During his last investigation, Mr. Holmes found a paper with some weird text.
He immediately recognized Cesar's Cipher, but he's wondering if you're really worthy of being his right hand man and is willing to take a chance on you. You will have to write the function that applies Cesar's Cipher to a character.
The message is only composed of upper case letters.

```csharp
public static char CaesarChar(char c, uint n);

CaesarChar('A', 2) == 'C'
CaesarChar('Z', 3) == 'C'
CaesarChar('A', 28) == 'C'
CaesarChar('R', 5) == 'W'
```

> **Hint**
>
> For more details, feel free to check **this article**

> **Clue**
>
> 32D<6CG:==6-15

There is nothing more deceptive than an obvious fact.