TP C#1 :

1 Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante :

```
rendu-tpcs1-login_x.zip
|-- rendu-tpcs1-login_x/
    |-- AUTHORS
    |-- hello.sln
    |-- hello
        |-- Tout sauf bin/ et obj/
    |-- lipogramme.sln
    |-- lipogramme
    |-- myCalculatrice.sln
    |-- myCalculatrice
    |-- printMe.sln
    |-- printMe
```

Bien entendu, vous devez remplacer "`login_x`" par votre propre login.

N'oubliez pas de vérifier les points suivants avant de rendre :

- Le fichier `AUTHORS` doit être au format habituel (une *, une espace, votre login et un retour à la ligne).
- Pas de dossiers `bin` ou `obj` dans le projet.
- **Le code doit compiler !**

2 Introduction

2.1 Objectifs

Pour ce TP, la rigolade c'est fini (ou presque). Le C# est désormais officiellement votre arme de guerre et nous allons pouvoir commencer les choses sérieuses. Rappelez-vous que c'est dans ce langage que vous ferez (très probablement) votre projet. Il est donc nécessaire de parfaitement comprendre les quelques bases qui vont vous être enseignées ici.

Nous aborderons dans ce TP des notions de base, qui ne devraient pas énormément vous dépayser de ce que vous connaissez déjà, mais qui permettront de vous habituer à la syntaxe du C#.

La majorité des fonctions que vous devrez coder ici sont des grands classiques. Elles permettent de voir les bases d'un langage sans avoir à implémenter des algorithmes trop complexes et vous permettront de prendre en main le langage. Si ces fonctions vous posent des problèmes, résolvez les en posant des questions, car vous y aurez droit tout au long de votre scolarité à l'Épita.

3 Cours

3.1 L'impératif

Le C#, est un langage impératif, à l'inverse de notre bon ami Ocaml. Mais concrètement, quelles sont les différences entre langages fonctionnels et impératif ?

Afin de répondre à cette question, utilisons la source fiable par excellence, Wikipédia :

> « La programmation impérative est un paradigme de programmation qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. Ce type de programmation est le plus répandu parmi l'ensemble des langages de programmation existants, et se différencie de la programmation déclarative. »

Ainsi, parmi les différences notables entre Ocaml et le C#, on trouve la disparition des identifiants au profit des variables, ainsi que l'apparition des boucles (que nous aborderons dans un prochain TP).

3.2 Les variables

En Ocaml, lorsque l'on souhaite stocker une donnée, qu'elle soit un caractère, un entier, ou quoi que ce soit, nous sommes obligés de déclarer un identifiant. Par nature cet identifiant diffère d'une variable par le fait qu'il ne peut être changé durant l'exécution du programme.

Le C# quant à lui vous ouvre de nouveaux horizons : une fois une variable déclarée, vous avez la possibilité de la changer n'importe quand durant l'exécution du programme. Tout comme en Ocaml, une variable est définie par trois choses qui nous intéressent ici : son type, son nom et sa valeur. À l'inverse de l'Ocaml, le type d'une variable est défini lors de l'écriture du programme, et son type est fixe. La déclaration d'une variable se fait ainsi :

```
TYPE name = VALUE;
```

Voici les types les plus courants que vous rencontrerez :

```
int my_int = -42;                          // Entier
uint my_uint = 42;                         // Entier non signe
string my_string = "Hello World!";         // Chaine de caracteres
char my_char = 'C';                        // Caractere
bool my_bool = true;                       // Booleen
float my_float = -1337.42;                 // Nombre a virgule flottante
```

3.3 Branchements conditionnels

Les branchements conditionnels font partie des composants principaux de tout programme, quel que soit le langage dont il est question. C'est notamment ce qui fait la différence entre un langage de balisage (HTML..) et un langage de programmation.

Il existe de nombreuses manières de faire des branchements conditionnels en C#. Parmi ceux-ci, nous trouvons notamment les if, ainsi que les switch, que nous traiterons ici [1].

1. Le goto est un branchement non conditionnel, mais il n'est cependant pas autorisé, et est déconseillé d'une manière générale. Rien ne vous empêche de vous renseigner dessus pour votre culture générale

```
if ( /*Condition*/ )
{
        // Instructions
}
else if ( /*Condition*/ )
{
        // Instructions
}
/* Autres else if*/
else
{
        // Instructions
}
```

À noter que le code ci-dessus montre l'une des nombreuses possibilités de combinaisons de ce branchement. En réalité, vous n'êtes pas obligés d'ajouter un else (if), et vous pouvez mettre autant de else if que vous le souhaitez. Dans tous les cas, seul l'un des branchements sera activé (le premier dont le test est réussi).

Dans le cas du if les conditions utilisent les opérateurs de comparaison suivants :

```
        ==              // Equal
        !=              // Different
        <               // Inferieur
        >               // Superieur
        <=              // Inferieur ou egal
        >=              // Superieur ou egal
```

Ces opérateurs permettent de renvoyer un booléen, que vous pouvez donc récupérer dans une variable de type bool. Vous pouvez bien entendu ensuite utiliser les opérateurs logiques suivants avec le retour de vos expressions.

```
        &&              // ET logique
        ||              // OU logique
        !               // NON logique
```

Notez aussi la présence dans cet exemple de deux types de commentaires, l'un commentant le reste de la ligne (//), et l'autre permettant de limiter le commentaire en longueur, et sur plusieurs lignes

Dans le cas où vous avez un grand nombre de valeurs possibles pour une même variable, avec une action différente pour chaque, l'utilisation d'un switch est fortement conseillée :

```
switch (var)
{
        case "A":
                // Instructions
        break;

        case "B":
                // Instructions
        break;

        case "C":
```

```
                // Instructions
        break;

        default: // Aucun case n"a abouti
                // Instruction
        break

}
```

À noter que vous ne pouvez pas utiliser d'opérateurs de comparaison dans l'utilisation d'un switch.

3.4 Fonctions

Une fonction est un morceau logique de votre programme. Tout comme en Ocaml, il vous est conseillé de segmenter votre code en plusieurs fonctions, chacune représentant un ensemble logique cohérent.

Une fonction en C# est définie par son prototype. Celui-ci est défini par la personne écrivant le programme et indique explicitement le type de retour, le nom de la fonction ainsi que les paramètres requis. Vous noterez aussi dans l'exemple ci-dessous la présence des mots clés "public" et "static", dont nous nous préoccuperons plus tard, lorsque nous nous pencherons sur les notions de Programmation Orientée Objet :

```
        public static typeRetour NOM(TYPE arg1, TYPE arg2)
        {
                // Instructions
        }
```

`typeRetour` se classe en deux catégories, selon ce que vous souhaitez faire :

- `typeRetour` est un type du langage, que ce soit `char`, `string`, `int`, auquel cas, votre fonction renverra évidemment le type que vous avez choisi, afin que vous puissiez l'utiliser dans une variable, une autre fonction, etc.
- Vous souhaitez faire une fonction qui ne retourne rien, auquel cas, vous remplacez `typeRetour` par `void`[2]. Une telle fonction peut être utile par exemple dans le cas où vous voulez uniquement afficher quelque chose sur la console...

`NOM` se réfère au nom de fonction que vous avez choisi. Son choix peut paraitre anodin, mais il vaut mieux passer deux minutes à réfléchir sur le nom à donner à une fonction, plutôt que de devoir relire tout son code chaque fois qu'il est nécessaire d'effectuer une modification sur le code.

Enfin, chaque argument donné à la fonction doit être séparé par une virgule. Vous pouvez en mettre autant que vous voulez, y compris aucun[3]. Chaque argument est défini comme toute variable, en indiquant son type ainsi que son nom.

2. Equivalent du unit() de Ocaml
3. À noter que si vous avez besoin de trop d'arguments, vous avez certainement un problème de conception dans votre code, il serait temps d'y réfléchir

Un petit exemple pour mettre tout ça en lumière :

```
public static int the_answer(int nb1, int nb2)
{
        int first_result = nb1 + nb2;
        first_result += 4; // On additionne un nombre a first_result
        first_result -= nb1;    // On soustrait un nombre a first_result
        first_result -= nb2;
        int result = first_result * 10 + 2;

        return result; // On retourne notre resultat de type int
}

public static void print_fun()
{
        System.Console.WriteLine("Hello World!");
}

public static void print_sad()
{
        System.Console.WriteLine("GoodBye World!");
}


/* La fonction Main est une fonction particuliere, qui est appelee
en premier au lancement du programme*/
static void Main()
{
        int foo = first_result(1337, 666);
        if (foo != 42)
        {
                print_sad();
        }
        else
        {
                print_fun();
        }
}
```

Essayez ce code, et déduisez-en le niveau intellectuel de la personne qui a écrit cet exemple. Libre à vous de mentionner cet avis dans le README, au risque de blesser l'auteur du sujet.

4 Exercices

4.1 Avant de commencer

Au cours de ces exercices vous devrez souvent afficher du texte en console, sans indication contraire tout affichage doit se terminer par un retour à la ligne.
Vous remarquerez aussi que si vous lancez votre programme avec Visual Studio, il va apparaitre et disparaitre bien trop rapidement pour que vous puissiez y voir quoi que ce soit. Pour remédier à ce problème vous pouvez soit ajouter un appel à `Console.Read()` à la fin de votre `Main` afin de rester ouvert en attendant une entrée de l'utilisateur ou simplement lancer votre programme depuis une invite de commande Windows.

Il est temps de rentrer dans le vif du sujet, créez un nouveau projet console dans Visual

Studio en faisant File/New/Project/Console Application avec pour nom "tpcs01" et lancez-vous dans les exercices suivants !

4.2 Exercice 0 : Un peu de lecture

Durant ce TP, vous allez être amenés à utiliser les chaines de caractères, aussi appelées `String`. La documentation d'un langage étant sensée devenir votre bible, nous allons ici vous forcer à en lire un petit peu. Aussi, avant de demander à vos ACDC comment fonctionne telle ou telle fonction, nous vous invitons à chercher dans la documentation de référence du C#, msdn.

Dans notre cas, vous allez devoir lire la page de présentation des `String` du msdn [4]. Merci de lire jusqu'à la sous partie "Accessing Individual Characters". Nous vous conseillons aussi de vous renseigner sur les fonctions utilisées dans les exemples, notamment les fonctions System.Console.Write ainsi que System.Console.ReadLine.

Il va sans dire qu'utiliser une fonction qui fait directement l'action demandée par un exercice sera considéré comme de la triche (par exemple les fonctions mathématiques), et sera sévèrement sanctionné. En cas de doute, demandez à vos ACDC.

4.3 Exercice 1 : Hello !

Pour cet exercice, nous vous demandons d'écrire une fonction avec le prototype suivant :

```
static void SayHello(string target);
```

Cette fonction doit afficher sur la console "Hello", suivi d'un espace, du contenu de la variable `target`, suivi directement d'un point d'exclamation, puis d'un retour à la ligne. Ajoutez dans votre Main() l'appel de cette fonction avec "World", "ACDC" et "Mme Cavatorta". Vous devriez avoir un resultat comme celui-ci :

```
Hello World!
Hello ACDC!
Hello Mme Cavatorta!
```

Ces tests doivent être conservés dans votre fonction Main() et être rendus avec. Votre fonction sera placée dans un projet nommé "hello"

4.4 Exercice 2 : Lipogramme

Le lipogramme est une figure de style qui consiste à produire un texte dans lequel une lettre donnée n'est pas utilisée. Vous l'aurez compris, votre objectif ici va être de vérifier qu'une chaine de caractères est un lipogramme. Pour cela, les boucles vous sont interdites, vous devrez le faire de manière récursive.

N'ayant pas encore vu de fonction récursive en C#, en voici un magnifique exemple inutile :

```
bool useless(int accu)
{
        if (accu > 0)
        {
                return useless(accu - 1);
```

4. http ://msdn.microsoft.com/en-us/library/ms228362.aspx

```
                }
                return true;
        }
```

Ici, vous allez devoir écrire une fonction avec le prototype suivant :

```
        static bool lipogramme(string str);
```

Cette fonction renvoie true si str est un lipogramme de "e" et "E", et false sinon. Afin de parcourir la chaine récursivement nous vous conseillons d'utiliser une fonction annexe contenant un index que vous incrémenterez à chaque appel. Rendez cet exercice en créant un projet nommé "lipogramme".

4.5 Exercice 3 : Calculatrice

Pour cet exercice, créez un nouveau projet nommé "myCalculatrice" [5]. Ensuite, implémentez les fonctions suivantes, qui vous suivront, comme précédemment dit, tout au long de votre scolarité.

4.5.1 Exercice 3.1 : myPow

Cette fonction doit renvoyer la n-ième puissance de x.

```
        static int my_pow(int x, int n);
```

4.5.2 Exercice 3.2 : myFact

Cette fonction renvoie la factorielle de n.

```
        static int my_fact(int n);
```

4.5.3 Exercice 3.3 : myFibo

Cette fonction retourne le n-ieme élément de la suite de fibonacci

```
        static int my_fibo(int n);
```

4.5.4 Exercice 3.4 : myPgcd

Cette fonction renvoie le PGCD entre a et b [6].

```
        static int my_pgcd(int a, int b);
```

4.5.5 Bonus

Écrire une fonction qui lit sur la console la fonction à exécuter, puis demande les arguments nécessaires, puis affiche le résultat de la fonction demandée sur la console. Vous pouvez aussi vous renseigner sur la fonction ackermann et l'implémenter en tant que nouvelle fonction.

5. Sans les guillemets évidemment
6. Methode d'Euclide...

4.6 Exercice 4 : Print me !

Nous allons maintenant vérifier que vous maitrisez les fonctions d'affichage de la console. Pour cela, nous vous demandons de dessiner une maison à plusieurs étages. Vous pouvez vous servir de tous les caractères contenus dans la table ASCII. Bien évidemment, nous serons plus regardants sur le style du code que sur la qualité du dessin. Cet exercice sera rendu dans un projet nommé "printMe".

4.6.1 Le toit !

Tout d'abord, nous aurons besoin d'un toit, avec option cheminée, celle-ci devra être dessinée en appelant la fonction :

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

En voici un exemple :

```
  /\
 /  \
/____\
```

4.6.2 L'étage.

C'est la partie faisant le plus appel à votre imagination, les plus acharnés utiliseront une fonction pour générer aléatoirement le motif pour ne pas répéter le même dessin de l'étage [7].

```
public static void floor();
```

Voilà à quoi il pourrait ressembler :

```
|                 |
|   /0\   /0\     |
```

Et la dernière partie de la maison...

Pour faire une maison entière, il ne vous reste plus qu'à dessiner les fondations dans la fonction suivante :

```
public static void ground();
```

Quelque chose qui ressemble à ceci :

```
|       _         |
|_____|-|______|
```

Il ne vous reste plus qu'à utiliser une fonction qui affichera les 3 à la suite et le tour est joué !

```
public static void house();
```

4.6.3 Bonus : une grande maison !

Bien évidemment, il serait dommage d'en rester là. Alors pourquoi ne pas afficher un nombre d'étages défini dans la console par l'utilisateur. Celle-ci déterminera le nombre d'étages à afficher.

```
public static void custom(int nb_floor);
```

The Bullshit is strong with this one

7. RTFM "Random"