TP C#1

1 Rendu

1.1 Archive

Vous devez rendre un .zip contenant l'architecture suivante :

```
– rendu-tp-prénom.nom.zip
    |– rendu-tp-prénom.nom/
        |– 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/
```

- Vous devez bien sûr remplacer *login* avec votre login (qui est *prénom.nom*).
- **Votre code doit compiler** (appuyer sur F5 doit afficher une console plutôt qu'une erreur).

1.2 AUTHORS

Ce fichier doit contenir : une étoile (*), un espace, votre login et un retour à la ligne.
Voici un exemple (où $ représente le retour à la ligne) :

```
* prénom.nom$
```

Le fichier doit s'appeler AUTHORS avec AUCUNE extension.

2 Introduction

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 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 davantage à la syntaxe du C#.

La majorité des méthodes 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 méthodes vous posent des problèmes, résolvez les en posant des questions, car vous y aurez droit tout au long de votre scolarité à l'EPITA.

Durant ce TP, nous verrons les notions suivantes :

- Les bases d'Arch Linux
- Le mode impératif
- Les variables
- Les conditions

3 Cours

3.1 Arch Linux

Vous avez sûrement déjà entendu parler du célèbre système d'exploitation qu'est Linux. Cela prendrait trop de temps de vous apprendre exactement ce qu'est Arch Linux donc retenez que c'est un "descendant" de Linux. Arch Linux est donc une distribution de Linux comme Debian, Ubuntu, Fedora, RedHat...

La philosophie de Linux est d'utiliser son terminal (la console noire des h4ck3rs) pour se déplacer dans son système et aller beaucoup plus vite qu'en cliquant avec votre souris.

Voici les commandes les plus basiques mais très utiles pour débuter avec Arch Linux :

- cd <nom du dossier> (Change Directory) - Changer de dossier
- ls (List) - Lister les fichiers du dossier courant
- mkdir <nom du dossier> (Make Directory) - Créer un dossier
- rm <nom du fichier> (Remove) - Supprimer un fichier
- touch <nom du fichier> - Créer un fichier
- man <commande> - THE programmer's manual: à utiliser sans modération, dès que vous voulez savoir ce que fait une commande

3.2 Le mode impératif

C# est un langage impératif, contrairement à notre bon ami Caml. Mais, quelles sont les différences entre langage fonctionnel et langage impératif?

Regardons la définition Wikipédia de programmation impérative :

« En informatique, 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 (dont la programmation logique ou encore la programmation fonctionnelle sont des sous-ensembles). »

Les principales différences entre Caml et C# c'est que l'on "remplace" les identificateurs par des variables, et l'apparition des boucles (que vous verrez dans un prochain TP).

Vous devez retenir que la programmation impérative c'est comme une recette: on exécute des instructions les unes après les autres. C'est la manière "naturelle" de programmer.

3.3 Variables

En C#, contrairement au Caml, vous devez donner à votre programme le type de la variable que vous allez déclarer. En effet, le C# est un langage fortement typé. Oubliez le "let", maintenant vous devez utiliser "int", "string" ou "float". Ci-dessous est un tableau résumant les principaux types et leurs descriptions.

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	Number simple precision from -3,402823e38 to 3,402823e38
double	Number double precision from -1,79769313486232e308 to 1,79769313486232e308
char	Character
string	A sequence of zero or more characters
bool	Has only two states : true and false

Voici un exemple pour utiliser des variables dans votre code :

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

Vous noterez que vous n'avez pas forcément besoin d'assigner une valeur à votre variable. Cela peut être utile dans certains cas, mais il faut toujours les initialiser avant de les utiliser sans quoi vous allez avoir d'étranges résultats. [1]

[1] Vous verrez ces cas dans les prochains TPs.

3.4 Conditions

Durant votre (long) voyage en tant que développeur, vous allez devoir vérifier si une variable est égale à quelque chose, si elle est plus grande ou non. Pour cela, vous utiliserez des conditions et des opérateurs. Vous allez principalement utiliser le très connu "if else" et les opérateurs suivants :

- a == b - a égal à b
- a != b - a différent de b
- a < b - a inférieur à b
- a <= b - a inférieur ou égal à b
- a > b - a supérieur à b
- a >= b - a supérieur ou égal à b

Voici un exemple:

```
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 }
```

Si vous voulez vérifier plusieurs conditions dans une seule expression vous pouvez utiliser les opérateurs suivants :

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

Un autre moyen de faire beaucoup de tests sur une variable est d'utiliser un switch :

```
1  int a = 1;
2  switch (a)
3  {
4     case 1:          //Vérifie si a est égal à 1
5        /* ... */
6        break;        //N'oubliez pas le break, sinon
7                      //ça va aller sur toutes les conditions
8     case 2:
9        /* ... */
10       break;
11    case 3:
12       /* ... */
13       break;
14 }
```

4 Exercices

Comme pour le TP 0, nous allons utiliser *MonoDevelop* pour les exercices suivants.

Pour chaque exercice, vous devez créer un projet. Sur *MonoDevelop*, les projets sont appelés solutions.

Pour créer un nouveau projet, cliquer sur "Fichier" en haut à gauche de la fenêtre de *MonoDevelop* et suivez les instructions suivantes:

- Nouveau
- Solution
- .NET dans le menu de gauche
- Projet console (soyez sûr de sélectionner C# comme langage)
- Donnez un nom à votre projet (le nom doit suivre l'architecture de rendu)
- Cliquer sur suivant

Appuyez sur F5 pour compiler and lancer votre projet.

5 Exercice 1: Indentation et commentaires

Créez un nouveau projet et appelez le "Indentation".

Dans cet exercice, vous allez apprendre l'indentation et l'utilisation des commentaires. Ces principes seront indispensables dans tous vos futurs projets de programmation. Nous sommes dans le merveilleux monde du mode impératif, et pour le rendre encore meilleur, vos méthodes doivent être propres et lisibles !
Ces règles vous seront expliquées par vos ACDC et vous devez les RESPECTER. Pour voir si vous avez bien compris ce qu'était l'indentation, voici un bout de code qu'on ne veut plus voir:

```
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 }
```

Votre mission est de rendre à ce code la beauté qu'il mérite.

Il existe 2 sortes de commentaires en C# (comme dans beaucoup d'autres langages de programmation).

```
1 // commentaire d'une ligne
2 /*
3 commentaire
4 d'un
5 bloc de lignes
6 */
```

Comme vous l'avez probablement vu, le code d'au-dessus ne compile pas. Commentez la ligne qui n'a pas de sens et faites en sorte que votre programme compile.

Bien sûr, dans la vie réelle, les commentaires sont là pour ajouter une explication à votre code et non pas pour commenter des lignes inutiles.

Remarque : Dans tous les TP C#, nous utiliserons la coding style conseillée par la MSDN[2], c'est celle qui est utilisée dans toute la librairie standard du langage et dans la grande majorité des projets écrits en C#. En bref, les identifiants de classes, méthodes, enum, structs sont nommés avec le format 'MyNamedThing' et les identifiants de variables, arguments, constantes, attributs avec le format 'myNamedThing'. Cette convention de nommage s'appelle le *CamelCase* (je vous laisse deviner pourquoi "Camel"...). Pour votre culture générale, dans d'autres langages comme le C, on utilisera plutôt le *snake_case* : my_function pour les fonctions et my_variable pour les variables.

[2]La Microsoft Developer Network (MSDN) est la section de l'entreprise Microsoft qui s'occupe de ses relations avec les développeurs.

6 Exercice 2: Jouons avec les types

Créez un nouveau projet et appelez le "PlayWithTypes". Après cet exercice vous serez un maître dans l'art de la manipulation des types et des choses qu'il faut ou qu'il ne faut pas faire. Pour cela, nous allons vous demander de créer de nombreuses petites méthodes basiques.

6.1 We need fact

Vous devez coder la très connue méthode factorielle. Elle prend un *int* en paramètre et retourne la factorielle de ce nombre.

```
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!

Une facile, vous devez coder la méthode puissance. Elle prend deux *int* (x et y) et renvoie x à la puissance 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 ...!

Parce qu'il faut toujours dire bonjour, vous allez le faire ! Votre méthode prend un string name et l'affiche précédé de "Hello".

```
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

Le titre dit tout: votre méthode doit afficher si le nombre donné en paramètre est pair ou impair.

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

6.5 Différents angles

Dans cet exercice, vous devez coder deux méthodes. La première prend un angle en degrés et renvoie son équivalent en radian. La deuxième fait l'opposé.

```
1 public static float DegToRad(float x);
2 // DegToRad(1) == 0,0174533
3 // DegToRad(53) == 0,925025
4
5 public static float RadToDeg(float x);
6 // RadToDeg(1F) == 57,2958F
7 // RadToDeg(5.5F) == 286,479F
```

7 Exercice 3: Notation polonaise

Créez un nouveau projet et appelez le "Polish".

Dans cet exercice, nous allons apprendre ce qu'est la notation polonaise ou préfixe d'une expression arithmétique.
Une expression en notation polonaise pour ajouter les nombres 1 et 2 s'écrit "+ 1 2" plutôt que la notation classique (notation infixe) "1 + 2". Un autre exemple avec des parenthèses pourrait être "(5 - 6) * 7" en notation classique est égal à "* (- 5 6) 7" en polonaise.

Le but est de coder une mini calculatrice polonaise.

La première méthode à coder est *Eval*. Cette méthode prend un opérateur et 2 valeurs en arguments et retourne le résultat.

```
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
```

Maintenant que nous avons notre méthode *Eval*, nous pouvons coder la méthode *Main*. Cette méthode va demander un opérateur (+, -, /, *, %) et 2 valeurs à l'utilisateur puis appeler la méthode *Eval* avec eux.

Tips : Vous pouvez utiliser *string Console.ReadLine()* pour lire les entrées de l'utilisateur et *int Convert.ToInt32(string value)* pour convertir une *string* en *int*.

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

Vous devez avoir le résultat suivant dans votre console en appuyant sur F5 :

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

/ 28 7 = 4
```

8 Exercice 4: Dessinons des maisons

Créez un nouveau projet et appelez le "PrintHouse". Savez-vous afficher sur la console? C'est ce que nous allons voir. Dans cet exercice, vous allez afficher une maison. Pour cela, vous avez besoin de quatre méthodes.

8.1 The roof

Affichez le toit de votre maison, en utilisant le prototype suivant :

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

C'est votre toit donc vous pouvez en faire ce que vous voulez mais il doit ressembler à un vrai toit. Voici un exemple :

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

8.2 The floor

Affichez un étage de votre maison, en utilisant le prototype suivant :

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

Pareil ici, faîtes ce que vous voulez mais soyez réalistes. Vous pouvez également créer différents étages et les afficher en utilisant la manière que vous voulez.

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

8.3 The ground

Affichez le rez-de-chaussé de votre maison, utilisez le prototype suivant :

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

Pareil ici, faîtes ce que vous voulez mais soyez réalistes.

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

8.4 Print everything

Bien, c'est la méthode finale. Vous devez implementez la méthode *PrintHouse* qui affiche votre maison en entière en utilisant les méthodes précédentes.

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

Le résultat final :

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

8.5 Bonii

Si vous avez fini tout le TP vous pouvez ajouter des bonii ! Ils peuvent être n'importe quoi comme, par exemple, différents étages pour votre maison. Soyez créatifs !

The Code is the Law.