TP C#2 :

1 Consignes de rendu

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

```
rendu-tpcs2-login_x.zip
|-- rendu-tpcs2-login_x/
    |-- AUTHORS
    |-- tpcs2.sln
    |-- tpcs2
        |-- Tout sauf bin/ et obj/
```

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

Les deux dernières semaines, vous avez eu l'occasion de découvrir le monde merveilleux du C# . Aujourd'hui nous allons essayer de vous familiariser avec la console. Nous allons vous apprendre à gérer les entrées utilisateurs, ainsi que la sortie standard et la sortie d'erreur.

Dans ce TP, vous allez également manipuler des boucles en tout genre, car c'est une notion incontournable de la programmation impérative. Vous apprendrez à utiliser convenablement cette notion à travers plusieurs exercices liés à des fonctions mathématiques, et d'autres liés à la manipulation de "string" (des chaînes de caractères pour ceux qui auraient oublié).

Certaines fonctions vous rappelleront des souvenirs de l'époque de la récursivité, mais vous vous rendrez compte que la plupart de ces fonctions devront être pensé différemment lors d'une implémentation par boucles. Ce TP est là pour vous donner les bons réflexes à avoir.

3 Cours

3.1 La console

Avant d'arriver à EPITA, certains d'entre vous ont dû entendre parler du terminal. Mais Kezaco ?

> *A computer terminal is an electronic or electromechanical hardware device that is used for entering data into, and displaying data from, a computer or a computing system.*

Wikipedia

Comme vous l'aurez surement compris, le terminal est une interface homme/machine au même titre que le l'interface graphique et le combo clavier/souris. Le terminal qui nous intéresse est celui permettant d'envoyer et afficher des données sur un ordinateur.

Ainsi sous Windows, il suffit de chercher et exécuter cmd.exe afin de l'ouvrir. Vous pouvez aussi utiliser le PowerShell, aussi présent dans Windows, qui n'est autre que le successeur de cmd.exe. Nous ne nous attarderons pas plus sur le PowerShell ou l'utilisation de cmd ici. Les plus curieux d'entre vous sont bien entendu encouragés à se renseigner dessus. À présent voyons comment intéragir avec la console en C# .

3.2 Utilisation en C#

La manipulation de la console en C# s'effectue au travers de la classe. . . Console !
Sa documentation est disponible dans la bible du C# : MSDN
. Ce site sera pour l'année à venir votre meilleur ami. À utiliser sans modération !ch Vous pouvez y accéder via Google ou à l'adresse suivante : *http ://msdn.microsoft.com/en-us/library/system.console*

Avant de vous jeter à corps perdu dans ce TP, nous vous conseillons de chercher, notamment sur MSDN, des informations sur les fonctions suivantes :

1. Console.Write : utiliser écrire sur la sortie standard
2. Console.WriteLine : utiliser pour écrire sur la sortie standard et retourner à la ligne
3. Console.Read : utiliser pour lire caractère par caractère sur l'entrée standard jusqu'à la détection d'un retour à la ligne
4. Console.ReadLine : utiliser pour lire sur l'entrée standard jusqu'à la détection d'un retour à la ligne

3.2.1 Stdin & Stdout : entrée standard et sortie standard

Certains se demandent surement qu'est ce que la sortie et l'entrée standard. Il s'agit ni plus ni moins que de flux, de 'canaux' dans lesquels le terminal va écrire ou lire. Il vous sera par ailleurs demandé durant ce TP d'écrire vos erreurs sur la sortie d'erreur. Et oui, il n'y a pas uniquement que les deux flux présenté plus tôt.

Ainsi pour écrire sur cette différente sortie, rien de plus simple :

```
//Ecriture sur la sortie standard
Console.WriteLine("No, I'm your father");
//Ecriture sur la sortie d'erreur
Console.Error.WriteLine("NOOOOOOOOOO...");
```

Ainsi le résultat sera :

```
C:\foo.exe
No, I'm your father
NOOOOOOOOOO ...
```

Une chose intéressante est que l'on peut passer des arguments à notre programme via le terminal. Pour cela jetons un coup d'oeil au main

```
public class Hello
{
  public static void Main(string[] args)
  {
    Console.WriteLine(''Hello, World!'');
    Console.WriteLine(''You entered the following {0}
    command line arguments: '', args.Length );
    for (int i=0; i < args.Length; i++)
    {
      Console.WriteLine(''{0}'', args[i]);
    }
  }
}
```

Ce bout de code contient plein de choses étranges qui vous seront expliqué lors des TP suivants (public, static,...). Comme vous pouvez le voir, la fonction Main (qui est le point d'entrée de notre programme) prend en paramètre un tableau de strings. Celui contient les paramètres qui seront passés au programme à l'exécution.

Ainsi on obient le résultat suivant :

```
C:\hello.exe A B C D
Hello, World!
You entered the following 4 command line arguments:
A
B
C
D
```

3.2.2 Obtenir des informations sur le terminal et le modifier

Nous vous conseillons pour la suite du TP de jeter un œil aux différentes caractéristiques suivantes :

1. Console.WindowHeight
2. Console.WindowWidth
3. Console.ReadKey
4. Console.BackgroungColor
5. Console.ForegroundColor
6. ConsoleColor

'ConsoleColor' est un type couleur. Vous pourrez vous en servir pour modifier les couleurs du terminal (couleur de la police, couleur du fond, ...) Par ailleurs, les autres attributs présents sont des attributs modifiables ou non, vous permettant d'obtenir des informations sur le terminal. Ainsi la hauteur de celui-ci sera égale à 'Console.WindowHeight'.

Enfin pour finir cette partie cours et conseil pour le TP, voilà une petite aide pour le dernier exercice. Le bout de code en dessous vous permettra en effet de récupérer une touche entrée par l'utilisateur et de tester sa valeur.

```
public class testKey
{
  Console.WriteLine("Press a random key: ");
  public static void Main(string[] args)
  {
    ConsoleKeyInfo input = Console.ReadKey();
    if (input.Key.Equals(ConsoleKey.A))
    {

      Console.Clear();
      Console.WriteLine("You entered the A key");
    }

    Console.Clear();
    Console.WriteLine("You did not press the A key");
  }

}
```

Vous pouvez noter par ailleurs l'utilisation de la fonction 'Console.Clear' qui permet de nettoyer la console.

```
Press a random key:
```

Ainsi, si l'on presse la touche A, on aura :

```
You entered the A key
```

sinon :

```
You did not press the A key
```

Dans les deux cas, on se retrouve seulement avec les phrases finales étant donné que l'on "clear" la console avant d'afficher.

3.2.3 Les boucles

Contrairement à OCaml [1] où vous étiez limité à la récursion, en C# vous aurez le droit d'utiliser les boucles.

Le principe de la boucle est assez simple. Il s'agit d'émettre une ou plusieurs conditions et de réitérer une suite d'instruction tant que les conditions ne sont pas fausses.

La boucle for :

Voici la structure d'une boucle for :

```
for (valeur de depart; condition; action)
{
  instruction 1;
  instruction 2;
  ...
}
```

Pour ceux pour qui l'explication ci-dessus pourrait sembler un peu obscure, voici un exemple :

```
for (int compteur = 0; compteur < 50; compteur++)
{
  Console.WriteLine(``Hello World!'');
}
```

Ce bout de code va ainsi écrire sur le terminal "Hello World !" 50 fois. Rien de bien compliqué. Un peu plus dur maintenant !

```
for (int compteur = 100; compteur > 0; compteur -= 2)
{
  Console.WriteLine(``Hello, World!'');
}
```

Vous allez rire mais ce code ... fait exactement la même chose que le précédent. Pourquoi ? Et bien dans le second cas, on part de 'compteur = 100', puis on décrémente de 2 notre compteur à chaque tour de boucle tant que 'compteur > 0'. Or 50 * -2 = -100. On va donc bien écrire sur le terminal "Hello World" 50 fois.

La boucle while : La boucle while est une autre boucle présente donc la syntaxe est "plus simple" que celle du for.

```
while (condition(s))
{
  instruction 1;
  instruction 2;
  instruction 3;
  ...;
}
```

Ainsi, si l'on souhaite refaire le "Hello World !" à l'aide d'une boucle while, on obtient :

```
int i = 0;
while (i < 50)
{
  Console.WriteLine(``Hello World!'');
  i++;
}
```

1. Attention, cela ne veut pas dire qu'il n'y avait pas de boucles de OCaml . . . C'est juste que vous n'aviez pas le droit de les utiliser

Une autre variante de la boucle 'while' est la boucle 'do...while'.

```
int i = 0;
do
{
  Console.WriteLine(''Hello World!'');
  i++;
}while(i < 50)
```

La différence avec la boucle While est que le programme entrera au moins une fois dans la boucle. Pour 'while' et 'do while', attention à ne pas oublier l'incrémentation ... Sinon boucle infinie...

Pour terminer sur les différents types de boucle, sachez qu'il en existe un troisième appelé 'for each'. Encore une fois, les plus curieux sont encouragés à faire leurs recherches, mais elle ne vous sera d'aucune utilité pour l'instant.

Break, Continue, Return :

Les mots clés que nous allons vous présenter sont très important à connaître. Le 'break' est un mot clé qui indique au programme qu'il doit sortir de la boucle dans la quelle il se trouve actuellement.

```
for (int i = 0; i < 50; i++)
{
  for (int j = 0; j < 25; j++)
  {
    Console.WriteLine(''j is '' + j);
    if (j ==  5)
       break;
  }
  Console.WriteLine(''i is '' + i);
    if (i ==  0)
      break;
}
```

Le terminal affichera alors

```
j is 0
j is 1
j is 2
j is 3
j is 4
j is 5
i is 0
```

Le mot clé 'continue' quant à lui permettra de "sauter" un tour de boucle et passer au suivant.

```
for (int i = 0; i < 5; i++)
{
  if (i % 2 != 0)
     continue;
  Console.WriteLine(''i is '' + i);
}
```

Le terminal affichera alors

```
i is 0
i is 2
i is 4
```

Enfin le mot clé 'return' [2] sortira des boucles mais aussi de la fonction. À utiliser avec prudence donc.

```
  void main()
  {
    int i = funct();
    Console.WriteLine(``i is '' + i);
  }

  int funct()
{
  for (int i = 0; i < 5; i++)
  {

    if (i % 2 != 0)
    return i;
  }
   return i + 10;
}
```

Le résultat est ainsi :

```
i is 0
```

2. (Il s'agit du fameux return débranchant cf Junior...Si elle vous demande, on ne s'est jamais vu...).

4 Exercices

4.1 Le début des festivités

Maintenant que vous avez tous parfaitement compris ce qu'était la console et que les boucles sont devenues votre passion dans la vie, nous allons pouvoir commencer ce TP. Les fonctions que vous allez devoir implémenter dans un premier temps sont des fonctions simples utilisant des notions incontournables de la gestion Entrée/Sortie de la console. Apprenez à bien les utiliser et posez un maximum de questions si vous ne comprenez pas ces notions car elles représentent la base de tout.

Comme on sait que vous adorez les maths, la deuxième partie est consacrée à l'implémentation d'un certain nombre de fonctions mathématiques en utilisant les boucles. Ici encore, posez des questions car ces exercices vous seront redemandés très souvent au fil de votre scolarité à l'EPITA.

Pour finir ce TP en beauté, nous vous demanderons de créer des fonctions récapitulant tout ce que vous avez pu apprendre sur les boucles et la console au long de ce TP. Ces deux derniers exercices sont conséquents alors prenez le temps de bien faire les choses, et faîtes votre maximum.

4.2 Exercice 0 : Mise en jambes

4.2.1 Exercice 0.1 : Who are you ? (who who, who who...)

La semaine dernière, vous avez commencé à manipuler la console, et celle-ci ne présente désormais plus aucun mystère pour vous. Histoire de démarrer en douceur et de vérifier que vous avez bien compris comment utiliser les entrées et sorties utilisateurs, nous allons vous demander de créer une fonction très simple qui demandera à l'utilisateur de rentrer son nom, et qui lui répondra "Bonjour" suivi du prénom entré par l'utilisateur.

Voilà à quoi devra ressembler votre programme une fois compilé :

```
> Please, enter your name
Brian
> Hi Brian!
```

```
static void who_are_you();
```

4.2.2 Exercice 0.2 : Pair ou impair

Cette fonction devra prendre en paramètre un entier n, si ce dernier est pair, elle devra écrire "Your number is even" sur la sortie standard, sinon, elle devra écrire "Your number is odd" sur la sortie d'erreur.

```
static void even_or_odd();
```

4.3 Exercice 1 : Le nombre secret

Pour cet exercice, vous devrez créer une fonction qui prendra en paramètre un entier sup, et devra faire deviner à l'utilisateur un nombre choisi aléatoirement entre 0 et sup.

Voilà à quoi elle devra ressembler :

```
> You have to find a number between 0 and 10
> You think that the number is...?
7
> Lower!
5
> Greater!
6
> Congratulations! The secret number is 6 (found after 3 attempts)
```

Et le prototype :

```
static void greater_or_lower(int sup);
```

4.4 Exercice 2 : Mais qui a touché à System.Math ? !

Malheur ! Alors que vous alliez tranquillement faire votre TP de la semaine (parce que vous êtes tous des élèves très sérieux et que vous ne voudriez pas mettre en colère vos ACDC adorés) vous vous rendez compte que quelqu'un a complètement effacé la bibliothèque Math !

Vous en comprenez les conséquences, vous n'avez plus accès aux fonctions mathématiques de la bibliothèque System C# . Mais ce n'est pas grave, vous vouez un véritable culte aux mathématiques et vous décidez donc de créer vos propres fonctions pour remplacer la bibliothèque disparue.

Il est évident que pour l'intégralité de cet exercice, vous n'avez pas le droit d'utiliser les fonctions de 'System.Math'.

4.4.1 Exercice 2.1 : my_multiplication

Première fonction à implémenter : la multiplication de nombres positifs. Oui, elle ne fait pas partie de 'System.Math' mais comme vous aimez beaucoup les maths, vous décidez d'en faire un peu plus. **Bien évidemment l'opérateur '*' n'est pas autorisé.**

```
static int my_multiplication(uint a, uint b);
```

4.4.2 Exercice 2.2 : my_eucl_div

Maintenant que vous avez fini la multiplication, vous vous attaquez à la division euclidienne, toujours sur des entiers positifs. **Ici les opérateurs '/' et '%' sont interdits.**

4.4.3 Exercice 2.3 : my_pow

Vous devez maintenant créer la fonction my_pow qui élève un nombre à une puissance donnée. La fonction sera testée avec des très grands nombres et renverra donc un long.

```
static long my_pow(int n, int pow);
```

4.4.4 Exercice 2.4 : my_abs

La fonction abs renvoie comme vous vous en doutez la valeur absolue d'un réel passé en paramètre

```
static float my_abs(float f);
```

4.4.5 Exercice 2.5 : my_sqrt

C'est ici que les choses se gâtent. Vous devez créer la fonction my_sqrt qui comme son nom l'indique renvoie la racine carrée d'un entier passé en paramètre. Cependant, il serait trop compliqué d'obtenir la valeur exacte de cette racine carrée, c'est pourquoi votre fonction devra renvoyer une approximation à 10^{-5} près. Vous utiliserez la méthode de Newton [3] pour déterminer cette valeur.

```
static float my_sqrt(int n);
```

4.4.6 Exercice 2.5 : my_fibo_iter

Comme vous l'aurez compris, vous devez implémenter la fonction fibonacci en itératif. Cette implémentation est plus compliquée que la version récursive naïve que vous avez pu voir en séminaire Caml, mais elle est bien plus rapide et surtout elle fonctionne avec des grands nombres.

```
static long my_fibo_iter(int n, int pow);
```

4.4.7 Exercice 2.6 : Jedi ou Sith ?

Après avoir passé les trois dernières minutes à faire les exercices précédents, vous recevez un coup de fil de votre ami Jedi qui vous assure avoir trouvé un garçon au potentiel incroyable. Il vous demande votre avis sur l'avenir de l'enfant, est-il voué à être un Sith ou un Jedi ?

Vous décidez donc de mettre les mathématiques et le C# au service de la force en créant un programme qui détermine de manière très simple l'avenir de n'importe qui. Votre fonction devra demander à l'utilisateur son nom, et déterminera son avenir de la manière suivante :

- Si le nombre de lettre dans son prénom est un nombre premier, c'est un Jedi
- Sinon, c'est un Sith

Vous l'aurez compris, l'important ici est l'implémentation de la fonction 'is_prime' qui vérifie si un nombre est premier.

```
static void jedi_or_sith();
```

3. http ://fr.wikipedia.org/wiki/Méthode_de_Newton

Et voilà le prototype de la fonction 'is_prime'

```
static bool is_prime(int n);
```

4.4.8 Exercice 2.6.1 : (Bonus)is_prime optimisé

En réalité, si vous implémentez une version naïve de 'is_prime', elle sera très longue sur des grands nombres. Il existe une méthode utilisant la racine carrée pour améliorer le temps d'exécution de cette fonction. En bonus, vous devrez donc améliorer votre 'is_prime' pour qu'il s'exécute rapidement sur des grands nombres, et votre fonction devra utiliser votre fonction 'my_sqrt'.

4.4.9 Exercice 2.7 : (Bonus) my_rotn

Vous avez passé du temps à faire vos recherches, vous pouvez désormais savoir qui est un Sith et qui est un Jedi de manière très simple, mais maintenant vous aimeriez beaucoup pouvoir garder vos informations pour vous. Pour cela, vous décidez de créer une fonction qui effectuera une rotation de rang n sur chaque caractère d'une string passée en argument. Par exemple, si votre rang est de 5 et que le caractère que vous traitez est 'd', vous devrez renvoyer le caractère 'i'. Attention, votre fonction devra être en mesure de gérer les lettres majuscules et minuscules et n'appliquera la rotation que sur les lettres, les caractères spéciaux ainsi que les chiffres ne seront pas modifiés.

```
static string my_rotn(int rank, string str);
```

4.5 Exercice 3 : Here comes the funny part

Si vous en êtes là, c'est que vous avez survécu à la partie mathématiques de notre TP. Félicitations ! Vous méritez bien un petit peu de changement, sortons des formules et amusons nous avec la console !

4.5.1 Exercice 3.1 : Je connais mon alphabet

Dans cet exercice, vous allez devoir implémenter une fonction qui affichera l'alphabet de a à g sur la console, suivi de la phrase "je connais mon alphabet". Facile ? Et bien pas tant que ça, puisque nous allons rajouter des petites restrictions :

- Vous ne devez pas utiliser la string "abcdefg"
- Chaque caractère doit être écrit avec une couleur différente
- Chaque caractère doit être écrit sur un fond de couleur différente
- A la fin de la fonction, les couleurs doivent être rétablies.

```
static void print_alpha()
```

4.5.2 Exercice 3.1.1 (Bonus) Récite moi l'alphabet !

Dans cette partie, nous allons utiliser un gadget de la bibliothèque 'System.Speech' du C# : la synthèse vocale. Ainsi, le but de ce bonus est de faire réciter l'alphabet à notre ordinateur. Vous devrez donc reprendre votre fonction précédente, et faire parler votre ordinateur à chaque

lettre, et pour la dernière phrase.

Nous allons vous montrer rapidement comment fonctionne la synthèse vocale. Tout d'abord, vous devez savoir que le C# est un langage objet, et que l'outil de synthèse vocale est lui même un objet. Pas de panique vous n'avez pas besoin de cette notion pour vous en servir, la preuve, vous vous servez bien de 'ConsoleColor' alors que c'est un objet.

Pour utiliser la synthèse vocale, vous allez devoir ajouter la bibliothèque 'System.Speech' à votre solution de projet. Pour cela, faîtes un clic-droit sur l'explorateur de solution, choisissez 'ajoutez une référence', cochez 'System.Speech', appliquez et fermez.

Vous pouvez maintenant inclure la synthèse vocale à votre projet en rajoutant la ligne suivante en haut de votre fichier Program.cs :

```
using System.Speech.Synthesis;
```

Voici comment déclarer et initialiser la synthèse vocale :

```
SpeechSynthesizer voice = new SpeechSynthesizer()
```

Et pour a faire parler, rien de plus simple que :

```
voice.Speak("Votre message ici!");
```

4.5.3 Exercice 3.2 : my_funky_vader

Voilà, vous êtes arrivé au dernier exercice de ce TP qui je vous l'accorde n'est pas des plus courts. Mais il aborde des notions importantes et nous devons insister dessus. Ainsi, si vous n'avez sauté aucun exercice et que vous lisez ces lignes, alors les boucles et la console n'ont sûrement plus de secrets pour vous, et c'est tant mieux puisque vous aurez besoin de toutes les notions abordées cette semaine pour venir à bout de cet exercice.

Vous allez devoir écrire une fonction qui affichera Dark Vador en ASCII art sur la console. Mais ce n'est pas tout, votre Dark Vador devra se déplacer sur la console à l'aide des touches ZQSD du clavier, et ce, sans sortir des limites de la console. On espère que vous avez bien lu le cours, puisque toutes les fonctions énoncées dans la partie cours vous seront utiles pour cet exercice.

Afin de vous alléger la tâche, nous vous fournissons la fonction print_vader, qui prend en argument le nombre d'espaces à afficher avant chaque ligne, et qui affiche Dark Vador en ASCII art sur la console.

```
static void print_vader(string space);
```

4.5.4 Exercice 3.2.1 : (Bonus) De la couleur !

Dans ce bonus, vous devrez faire en sorte que Dark Vador change de couleur à chaque déplacement.

Vous pouvez ajouter vos propres bonus si vous en avez, à conditions qu'ils soient expliqués dans un fichier README (**en ANGLAIS !**) qui se situera dans le dossier racine de votre rendu (avec le fichier AUTHORS).

The bullshit is strong with this one...