TP C#2 : La console et ses mystères

1 Consignes de rendu

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

```
tpcs02-login_x.zip
    |– tpcs02-login_x/
        |– AUTHORS
        |– README
        |– src/
            |– tpcs02.sln
            |– tpcs02/
                |– Tout sauf bin/ et obj/
```

Vous devez, bien entendu, remplacer login_x par votre propre login. Avant de rendre, n'oubliez pas de vérifier :

- Que le fichier AUTHORS est bien au format habituel (une *, un espace, votre login et un retour à la ligne, dans un fichier sans extension)
- Que vous avez bien supprimé les dossiers bin et obj
- **Que votre code compile**

2 Cours

2.1 La console

En tant qu'utilisateur de Windows, vous connaissez sûrement ce qu'on appelle les GUI (Graphic User Interface, ou interface graphique en français), c'est à dire de jolis écrans, avec pleins de boutons ou de curseur sur lesquels on peut cliquer ou qu'on peut faire glisser. Cependant, il existe un autre monde, beaucoup plus sombre (littéralement), plus rapide également, qui fera croire à toute personne extérieure que si vous l'utilisez, c'est parce que vous êtes un hacker confirmé : la console.

Il y a plusieurs dizaines d'années, les informaticiens n'avaient pour seul outil que cette console en ligne de commande. Même si aujourd'hui cette époque est révolue, la console continue à survivre malgré la prolifération des GUI, grâce à son efficacité.

A partir de l'année prochaine (ou dès maintenant si vous le souhaitez), vous allez utiliser comme système d'exploitation Unix. Vous verrez que dans ce dernier, l'interface en ligne de commande est encore extrêmement présente, et a été développé afin de pouvoir être utilisé quasiment sans limite. Même si la console de Windows est un peu moins performante que cette dernière, nous allons quand même en développer le fonctionnement.

2.1.1 L'invite de commande Windows

Chaque version de Windows (même Windows 10) possède une console héritée de l'époque de MS-DOS et accessible en laçant cmd.exe (avec Win + R par exemple).

Vous pouvez exécuter un bon nombre d'action, mais voici les plus courantes :
— `dir` : Liste les fichiers du dossier courant
— `cd <chemin d'accès du nouveau dossier>` : Change le dossier courant
— `copy <nom du fichier> <chemin d'accès du dossier où copier>` : Copie un fichier dans un autre dossier
— `move <nom du fichier> <chemin d'accès du dossier où déplacer>` : Déplace un fichier dans un autre dossier

Il est également possible de lancer n'importe quel programme exécutable. en tapant son nom si il est dans le dossier courant, ou son chemin d'accès si ce n'est pas le cas.

Cependant, cette interface est assez limitée et n'est aujourd'hui gardée que pour des soucis de compatibilité.

2.1.2 Le Powershell

Comme son nom l'indique (shell = interpréteur de commande), il s'agit de la nouvelle version de la console de Windows, apparue avec Windows 7.

Son fonctionnement se rapproche plus de celle d'Unix, avec des commandes au nom assez proche.

Les commandes expliquées précédement deviennent :
- `ls` : Liste les fichiers du dossier courant
- `cd <chemin d'accès du nouveau dossier>` : Change le dossier courant
- `cp <nom du fichier> <chemin d'accès du dossier où copier>` : Copie un fichier dans un autre dossier
- `mv <nom du fichier> <chemin d'accès du dossier où déplacer>` : Déplace un fichier dans un autre dossier

2.1.3 La console en C#

En C#, la manipulation de la console se fait grâce à la classe "`System.Console`". Afin de trouver un peu d'information dessus, nous vous demandons de jeter un oeil à la documentation du C#, fourni par Microsoft : MSDN. MSDN contient quasiment tout ce qu'il est possible de faire en C#, et comment utiliser les classes prédéfinies, comme "`System.Console`". Voici un lien direct vers la page de MSDN concernant la console : https ://msdn.microsoft.com/fr-fr/library/system.console

Nous vous demandons de regarder particulièrement les méthodes suivantes, nous allons les utiliser extrêmement souvent pendant ce TP :
- `Console.Write`
- `Console.WriteLine`

— `Console.Read`
— `Console.ReadLine`

2.2 La boucle while

Une boucle `while` s'écrit de la manière suivante :

```
1 while(condition)
2 {
3     instructions;
4 }
```

Tant que la condition est vraie, les instructions entre les accolades sont répétées en boucle. Par exemple, le code suivant va compter dans la console de 0 à 10.

```
1 int i = 0;
2 while(i <= 10)
3 {
4     Console.WriteLine(i);
5     i++;
6 }
```

La ligne "`i++`" permet d'incrémenter notre compteur. Cela revient au même que d'écrire "`i = i + 1`" ou encore "`i += 1`". Attention à ne pas oublier d'incrémenter (ou de décrémenter selon les cas) votre compteur pour ne pas créer des boucles infinies.

2.3 La fonction `main`

La fonction `main`, générée automatiquement à la création d'un nouveau projet, est la seule et unique fonction qui sera appelée quand vous exécuterez votre programme. Par conséquent, si vous voulez tester les autres fonctions que vous coderez, vous allez devoir les appeler depuis la fonction `main`.

3 Exercices

Toutes les fonctions de ce TP devront être codées dans un nouveau projet sous Visual Studio, de type “Console Application”, avec pour nom “tpcs02”. A chaque fois que allez afficher du texte dans la console, sauf indication contraire, vous devez terminer par un retour à ligne pour plus de clarté. Par défaut, si vous n’arrivez pas jusqu’à l’exercice 5, vous devrez appeler toutes vos fonctions les unes à la suite des autres, dans l’ordre, dans votre fonction `main`, en affichant à chaque fois la ligne `"Exercise X:"` avant. Attention, la console doit rester ouverte à la fin de l’exécution de toutes vos fonctions !

3.1 Exercice 1 : HelloWorld

Écrivez une fonction qui écrit dans la console la phrase “`Hello World!`” suivie d’un retour à la ligne.

Prototype :

```
1 static void Hello()
2 {
3         //FIXME
4 }
```

3.2 Exercice 2 : Echo

Écrivez une fonction qui lit la prochaine phrase (jusqu’à un retour à la ligne) que l’utilisateur entre dans la console puis qui la répète.

Prototype :

```
1 static void Echo()
2 {
3         //FIXME
4 }
```

Exemple :

```
Exercise 2:
Test

Test
```

3.3 Exercice 3 : Ma console

Écrivez une fonction qui écrit le mot ACDC en ASCII Art en rouge sur fond bleu. Vous changerez également le titre de la console en "This is my own console !". Pour changer la couleur du fond de la console et au texte, jetez un oeil aux propriétés `Console.BackgroundColor` et `Console.ForegroundColor` sur MSDN. Pour le titre, regardez `Console.Title`.

Bonus : Utilisez uniquement un seul `Console.Write` pour écrire l'ASCII Art.

Le ASCII Art en question :

```
                _____    _____      _____
     /\        / ____|  |  __ \    / ____|
    /  \      | |       | |  | |  | |
   / /\ \     | |       | |  | |  | |
  / ____ \    | |____   | |__| |  | |____
 /_/    \_\    \_____|  |_____/    \_____|
```

Prototype :

```
static void MyConsole()
{
    //FIXME
        Console.ResetColor();
}
```

Le `Console.ResetColor` à la fin permet de remettre aux valeurs par défaut les couleurs de la console.

3.4 Exercice 4 : Décompte final

Écrivez une fonction qui compte de 3 à 0, en utilisant une boucle `while`.

Prototype :

```
static void Countdown()
{
        //FIXME
}
```

Exemple :

```
3
2
1
0
```

3.5 Exercice 5 : De nombreuses options

Comme vous l'avez peut-être remarqué, la fonction `main` prend comme paramètre "`string[] args`" par défaut. Quand vous appelez votre .exe depuis la console, vous avez la possibilité de rajouter un ou plusieurs arguments à la suite du nom du .exe. Exemple :

```
monprogramme.exe arg1 arg2
```

Chaque mot est stocké dans une case du tableau de chaîne de caractères `args`, et passé en argument lors de l'appel de la fonction `main` à l'exécution du programme. On peut donc récupérer les chaînes de caractères données par l'utilisateur et les utiliser dans le programme.

On va se servir de ces arguments afin de sélectionner quelle fonction vous allez appeler, parmi tout celles que vous avez coder. Si vous appelez votre programme de la manière suivante :

```
tpcs2.exe 1
```

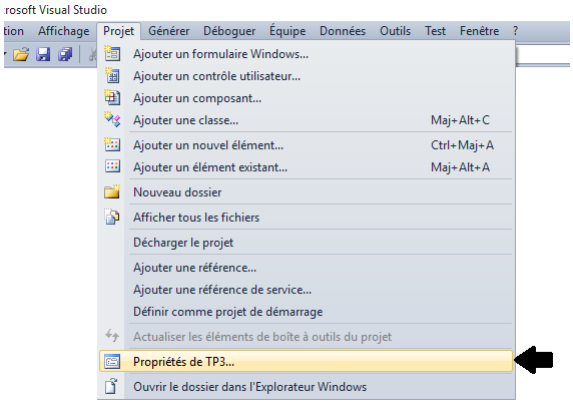
Vous allez lancer la fonction correspondant à l'exercice 1 du TP. Le fonctionnement des tableaux sera vu dans un prochain TP, et on se contentera d'utiliser la ligne de code suivante pour récupérer le premier argument passé à l'appel du programme, au début du `main` :

```
1 string s = args[0];
```

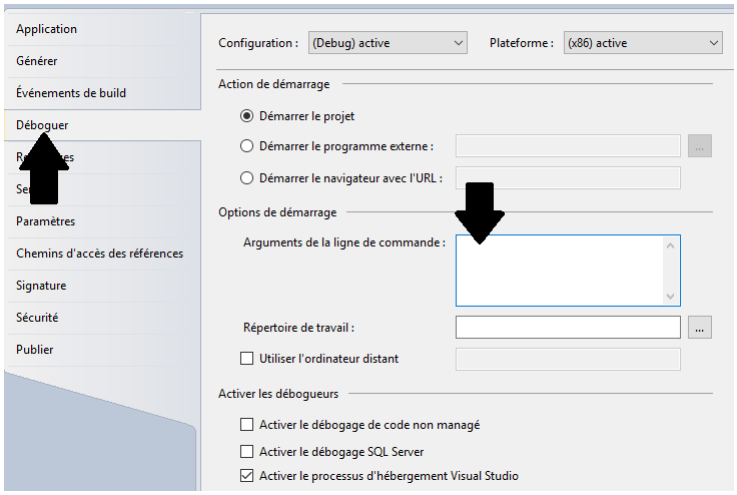
Il s'agira ensuite de comparer cette string pour savoir quelle fonction appeler.

Toutes vos fonctions devront être appelées avec une ligne `"Exercise X:"` écrite dans la console avant. Sauf indication contraire, l'argument devant être passé à votre programme pour appeler le nième exercise doit être juste le chiffre `n`. En cas d'argument ne correspondant à aucune fonction, vous devez écrire un message d'erreur dans la console. Attention, la console doit rester ouverte à la fin de l'exécution de la fonction choisie !

Pour tester votre programme sans avoir à le compiler puis à le lancer dans la console, vous pouvez directement avec Visual Studio lui donner des arguments. Pour cela, allez dans "Projet", "Propriété de (nom de votre fichier)...".



Puis, dans l'onglet qui s'est ouvert, choisissez "Déboguer" puis indiquez vos arguments dans la zone de texte "Argument de la ligne de commande". Par exemple, si vous voulez lancer l'exercice 1, mettez juste "1" dans la zone de texte, puis compilez votre projet.



3.6 Exercice 6 : Morse

Écrivez une fonction, qui à partir d'une string contenant uniquement les caractères '.', '-' et ' ', transmet le message donné en son.

Le '.' correspond à un bip de 150ms à 900Hz, le '-' à un bip de 450ms à 900Hz et ' ' à un silence de 450ms.

Vous pouvez utiliser `Console.Beep(f,n)` pour produire le son, avec pour paramètre `f` la fréquence en Hz et pour paramètre `n` la durée en milliseconde.
`System.Threading.Thread.Sleep(n)` permet de mettre le programme en pause pour `n` milliseconde.

Pour une string `s`, `s.Length` permet d'obtenir la longueur de la chaîne de caractères. `s[i]` renvoie lui le caractère d'index `i`. Attention, le premier caractère est à l'index 0 !

Exemple :

```
1 string s = "Test";
2 Console.WriteLine(s.Length);
3 Console.WriteLine(s[2]);
```

```
1 4
2 s
```

Cette fonction peut soit être récursive, soit utiliser une boucle `while`. Vous pouvez avoir besoin d'une seconde fonction si vous optez pour la récursivité.

Prototype :

```
1 static void Morse()
2 {
3         string s = ".... . .-.. .-.. ---  .-- --- .-. .-.. -..";
4         Console.WriteLine(s);
5     //FIXME
6 }
```

Bonus : Ecrire également la version complémentaire à celle que vous avez faite. Pour appeler cette fonction, l'utilisateur devra utiliser l'argument "6bonus" lors de l'appel au fichier .exe.

3.7 Exercice 7 : Sapins

Noël approchant à grand pas, le but de cet exercice va être de dessiner des sapins avec le caractère '*' dans la console. Attention, vous allez devoir générer vos sapins en fonction de la taille passée en paramètre de votre fonction en utilisant des boucles `while`. N'essayez pas de tricher en codant "en dur" l'affichage des sapins, leur présence sera vérifiée.

Prototype :

```
1 static void Pine(int n)
2 {
3     //FIXME
4 }
```

L'affichage se fait en deux parties : d'abord les branches, puis en suite le tronc. Pour les branches, vous aurez besoin de deux boucles `while` imbriquées (une pour aller de la branche `1` à `n`, une pour dessiner la branche en elle-même). De même, vous aurez besoin de deux autres boucles `while` imbriquées pour l'affichage du tronc, qui mesure toujours un caractère de large, et `n/3` de hauteur.

Pour l'appel dans le `main`, nous vous demandons de générer à la suite, un arbre de taille 3, un de taille 4 et un de taille 5.

Exemple :

```
Pine(3);
  *
 ***
*****
  *

Pine(5);
    *
   ***
  *****
 *******
*********
    *
```

Bonus : Rajoutez des boules de Noël à votre sapin (représentées par le caractère `'o'`), ainsi qu'une étoile (le `'x'`) en respectant le motif suivant :

```
Pine(5);
    x
    *
   *o*
  *o*o*
 *o*o*o*
*o*o*o*o*
    *
```

4 Bonii

4.1 Dessine moi un train

Ecrivez une fonction qui dessine en ASCII Art un train. Vous pouvez rajouter de la couleur si vous le voulez. Pour appeller cette fonction, l'utilisateur devra utiliser l'argument "trainbonus" lors de l'appel au fichier .exe.

Bonus : Animez votre train, rajoutez des nuages ou ce que vous voulez, tout est permis ! Vous pouvez vous inspirer du comportement de la commande sl sous Linux.

Prototype :

```
1 static void DrawTrain()
2 {
3         //FIXME
4 }
```