Fi-chiers !

Consignes

Dans ce TP, vous ne devez rendre **uniquement** que les fichiers sources (`.cs`) de vos programmes.
La structure du rendu doit être comme ceci :

```
/rendu-tp-login_x.zip
   |-- login_x /
          |-- AUTHORS
          |-- Basics.cs
          |-- ImageFilter.cs
          |-- README
```

`rendu-tp-login_x.zip`	nom de l'archive à rendre
`AUTHORS`	vous devez remplacer le contenu du fichier par une étoile `*`, un espace, puis votre login (ex : `login_x`) suivi d'un retour à la ligne
`README`	vous devez écrire dans ce fichier, tout commentaire sur le TP, votre travail ou bien plus généralement sur vos points forts/faibles. un fichier `README` vide sera considéré comme une archive sale (malus).

Il va de soi que toutes les occurrences de `login_x` sont à remplacer par **votre** login.

Tout le TP est à réaliser en mode console : `Fichier` ⇒ `Nouveau` ⇒ `Projet` ⇒ `Application Console`

TESTEZ VOTRE CODE !
Vous devez rendre un code qui compile, sans erreur ni warning !
Autrement votre rendu ne sera pas corrigé !

Introduction

Toutes les notions abordées dans cette section sont mises en pratique dans la partie annexes.

Les fichiers représentent le moyen le plus utilisé en informatique pour stocker des données. Votre système d'exploitation s'occupe de ranger ces fichiers dans votre ordinateur, sur le support physique approprié (disque dur, etc.). Ceci vous permet de simplement avoir à mémoriser un nom de fichier, et son chemin. On parle d'**arborescence**.
Plusieurs données sont attachées à un fichier, selon le système de fichiers (**filesytem**) il peut y avoir notamment la date à laquelle il a été créé, son propriétaire, son emplacement physique sur le disque dur, mais aussi son nom. On parle de **metadata** : des informations sur les informations.

D'autres données sont stockées dans le fichier lui-même (l'**inode**), en plus de "la" donnée essentielle. Citons par exemple : le titre et l'artiste des fichiers musicaux ou encore les dimensions et la date d'une photo.

Il existe deux types de formats de fichier, les fichiers *binaire* et les fichiers *texte* (ASCII). Les fichiers *texte* sont les fichiers les plus simples, ils ne contiennent que des caractères affichables (essentiellement ceux de votre clavier). Parmi les plus commun on trouve les fichiers de configuration ou les fichiers sources (`.cs`, `.xml`, ...). Attention ce n'est pas parqu'un fichier possède l'extension `.txt` qu'il s'agit d'un fichier texte, **ce n'est pas l'extension qui fait le fichier, et inversement**.

Pour lire et écrire dans un fichier en `C#`, nous utiliserons pour l'instant la classe `FileStream`. Cette dernière permet de manipuler les fichiers avec un système de flux (*Stream* en anglais). Les flux sont une représentation de données abstraites qui insiste sur le fait que les données peuvent être traitées à la file, avec une notion de consommation des données au fur et à mesure : quand je lis le premier octet d'un flux, si je recommence, j'obtiendrai l'octet suivant.

La classe `FileStream`, qui hérite de la classe `Stream` a bien ce fonctionnement, et propose en plus certaines fonctions propres aux fichiers. Pour une utilisation des fichiers texte, il existe les classes `StreamReader` et `StreamWrite`. Elles rajoutent notamment les méthodes de lecture et d'écriture par ligne (`ReadLine`, `WriteLine`) similaires à celle utilisées pour la sortie `Console`. Une sur-couche est également disponible pour la manipulation des fichiers *binaire*, il s'agit des classes `BinaryReader` pour la lecture et `BinaryWriter` pour l'écriture.

1 Mise en pratique

Fichier à rendre : `Basics.cs`

1.1 Lecture

Compléter la méthode `MyReader` de façon à afficher dans la console le contenu du fichier dont le nom est passé en paramètre. La méthode doit afficher des messages d'erreur explicites personnalisés en cas de problème (fichier introuvable, problème de droits, etc.).

```
1 static void MyReader(string filename) { /* FIXME */ }
```

1.2 Écriture

Compléter la méthode `MyWriter` de façon à recopier tout ce que l'utilisateur entre dans la console (jusqu'à une ligne vide) dans le fichier dont le nom est passé en paramètre. La méthode doit afficher des messages d'erreur explicites personnalisés en cas de problème (problème de droits, etc.).

```
1 static void MyWriter(string filename) { /* FIXME */ }
```

2 Des images

Fichier à rendre : `ImageFilter.cs`

Dans cette section le but va être de vous faire réaliser une petite classe de manipulation d'images. Chacune des méthodes décrites ci-dessous doit bien entendu appartenir à cette classe. Le modèle vous est redonné à la fin.

2.1 Introduction

The `BMP` file format, also known as bitmap image file or device independent bitmap (`DIB`) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.

– wikipedia

Le bitmap est donc un format non compressé d'image. Son principe est simple, le fichier commence par un `header` qui contient les informations relatives à l'image contenue, tels que ses dimensions, la couleur des différents pixels, etc.

La partie de ce `header` qui va nous intéresser suit le modèle suivant [1] :

signature	2 octets	ici ce sera "BM"
file size	4 octets	contient la taille du fichier.
reserved 1	2 octets	
reserved 2	2 octets	
offset	4 octets	contient la position où la description des pixels commence.
DIB header size	4 octets	
width	4 octets	contient la largeur de l'image en pixels.
height	4 octets	contient la hauteur de l'image en pixels.
planes	2 octets	
bits per pixel	2 octets	contient le nombre de bits par pixel

1. `http://en.wikipedia.org/wiki/BMP_file_format`

2.2 Informations

Compléter la méthode `ReadBMPInfos` de façon à ce qu'elle affiche dans la console les informations de la taille de l'image et du nombre de bits par pixel si la signature du `BMP` est correcte, un message d'erreur explicite sinon.

```
public static void ReadBMPInfos(string filename) { /* FIMXE */ }
```

2.3 Chargement / Sauvegarde

Vous allez devoir implémenter deux méthodes supplémentaires qui vont nous permettre de charger et sauvegarder notre image. Nous allons cependant utiliser pour cet exercice la classe `System.Drawing.Bitmap` [2] qui va nous permettre de manipuler beaucoup plus simplement nos images, et ceci indépendamment de leurs formats.

```
public bool LoadImageFrom(string filename) { /* FIXME */ }
public bool SaveImageTo(string filename) { /* FIXME */ }
```

2.4 Conversion

Maintenant que vous savez manipuler des fichiers et ce qu'est une image, nous allons pouvoir laisser la créativité parler, mais on va rester en mode console tout de même, nous n'avons donc pas d'autres choix que de se mettre à l'`ASCII Art` [3].

Le souci de l'ASCII Art c'est qu'une image de 1920x1080p ne peut pas tenir à l'échelle 1 :1 dans la console, il va vous falloir faire une correspondance entre nos pixels de notre image et les caractères correspondants dans la console. L'autre point gênant est qu'une image n'est pas forcément en noir et blanc, il vous faudra donc faire un passage en niveau de gris quelque part dans votre code [4].

Une fois ces soucis réglés, il va falloir associer à chaque groupe de pixels le caractère adéquate, nous allons pour cela utiliser les caractères suivant (du plus foncé au plus clair, de 0 à 255) :

```
private static char[] ascii =
    {'@','#','8','&','+','o',':','*','.',' '};
```

Compléter la fonction `AsciiCreator` de façon a ce qu'elle convertisse l'image, et stocke sa représentation en ASCII Art dans la chaîne de caractères passée en argument.

```
public void AsciiCreator(out string output) { /* FIXME */}
```

2. il peut être nécessaire de rajouter la référence `System.Drawing` à votre projet
3. `http://en.wikipedia.org/wiki/ASCII_art`
4. Conversion RGB/BW (30%R 59%G 11%B)

2.5 Filtres

Passons aux choses plus sérieuses, au lieu de faire de l'ASCII en console, nous souhaiterions plutôt sauvegarder les modifications que nous effectuons, et allons effectuer, sur nos images.

2.5.1 Grayscale

Compléter la méthode `Grayscale` de sorte qu'elle convertisse l'image en sa version en niveau de gris.

```
public void GrayScale() { /* FIXME */ }
```

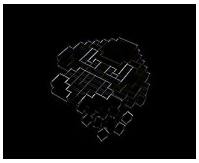
2.5.2 Processing

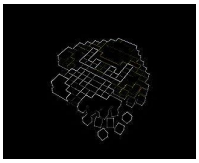
Dans cette partie vous allez devoir implémenter les différents filtres d'image listés ci dessous. Pour chacun de ces filtres reportez vous à la page Wikipédia [5] pour connaître les valeurs et l'effet de celui-ci. Soyez malin, il existe de grosses similitudes entre ces différents algorithmes, ceci sera pris en compte dans l'évaluation de votre code.

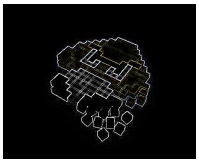
```
public void Identity() { /* FIXME */ }
public void Edge1() { /* FIXME */ }
public void Edge2() { /* FIXME */ }
public void Edge3() { /* FIXME */ }
public void Sharpen() { /* FIXME */ }
public void Blur() { /* FIXME */ }
public void Gaussian() { /* FIXME */ }
public void Unsharp() { /* FIXME */ }
```


input


identity


edge 1


edge 2


edge 3


sharpen


blur


gaussian


unsharpen

5. http://en.wikipedia.org/wiki/Kernel_(image_processing)

2.6 Boni

2.6.1 Annulation

Dans la hâte et la précipitation, nous avons pu appliquer trop de filtres successivement et donc ne pas obtenir le résultat escompté. Vous allez donc devoir implémenter les méthodes `Undo` et `Redo` qui permettent respectivement d'annuler et de restaurer la dernière modification appliquée. Attention il n'y a pas de limite au nombre d'annulation, restauration, possible. Il doit théoriquement être possible de revenir à l'image de départ.

```
1         public bool Undo() { /* FIXME */ }
2         public bool Redo() { /* FIXME */ }
```

2.6.2 Filtres (encore)

Bien évidemment les filtres proposés ici ne sont pas exhaustifs, vous avez donc la charge d'en implémenter d'autres et de les décrire dans le fichier `README`.

2.6.3 Interface

Il serait beaucoup plus facile d'avoir une petite interface graphique permettant de voir les résultats au fur et à mesure des applications...

2.7 Classe

```
1     class ImageFilter
2     {
3         private static char[] ascii =
4             {'@','#','8','&','+','o',':','*','.',' '};
5         public static void ReadBMPInfos(string filename)
6             { /* FIXME */ }
7
8         // no getter/setter allowed for attributes in this class
9         private Bitmap _data = null;
10        /* FIXME: we may need more attributes */
11
12        public ImageFilter() { /* FIXME */ }
13        public ImageFilter(string filename) { /* FIXME */ }
14
15        public bool LoadImageFrom(string filename) { /* FIXME */ }
16        public bool SaveImageTo(string filename) { /* FIXME */ }
17
18        public void AsciiCreator(out string output) { /* FIXME */}
19
20        public void GrayScale() { /* FIXME */ }
21
22        public void Identity() { /* FIXME */ }
23        public void Edge1() { /* FIXME */ }
24        public void Edge2() { /* FIXME */ }
25        public void Edge3() { /* FIXME */ }
26        public void Sharpen() { /* FIXME */ }
27        public void Blur() { /* FIXME */ }
28        public void Gaussian() { /* FIXME */ }
29        public void Unsharp() { /* FIXME */ }
30
31        public bool Undo() { /* FIXME */ }
32        public bool Redo() { /* FIXME */ }
33    }
```

Annexes

Indispensables

```
using System.IO;
using System.Drawing;
```

FileStream

```
FileStream f = new FileStream("test.txt",              // name
                              FileMode.Create,         // new file
                              FileAccess.ReadWrite);   // r+w

for (int i = 0; i < 26; ++i)
    f.WriteByte((byte)('A' + i));
f.Seek(0, SeekOrigin.Begin);                  // back to the start
Console.WriteLine((char)(f.ReadByte()));      // display the char 'A'
Console.WriteLine((char)(f.ReadByte()));      // display the char 'B'
f.Seek(1, SeekOrigin.Current);                // move 1 byte forward
Console.WriteLine((char)(f.ReadByte()));      // display the char 'D'

f.Close();                                    // ! mandatory close !
```

Stream

```
StreamReader sr = new StreamReader("test.txt");
string line = sr.ReadLine();
sr.Close();
```

```
FileStream f = new FileStream("test.txt", FileMode.Open,
                                          FileAccess.Read);
StreamReader sr = new StreamReader(f);
string line = sr.ReadLine();
sr.Close();
f.Close();
```

```
StreamReader sw = new StreamWriter("toto.txt");
sw.WriteLine("a beautiful sentence!");
sw.Close();
```

BinaryReader

```
FileStream f = new FileStream("toto.txt", FileMode.Open,
                                          FileAccess.Read);
BinaryReader br = new BinaryReader(f);
int i = br.ReadInt32();
char c = br.ReadChar();
f.Close();
```