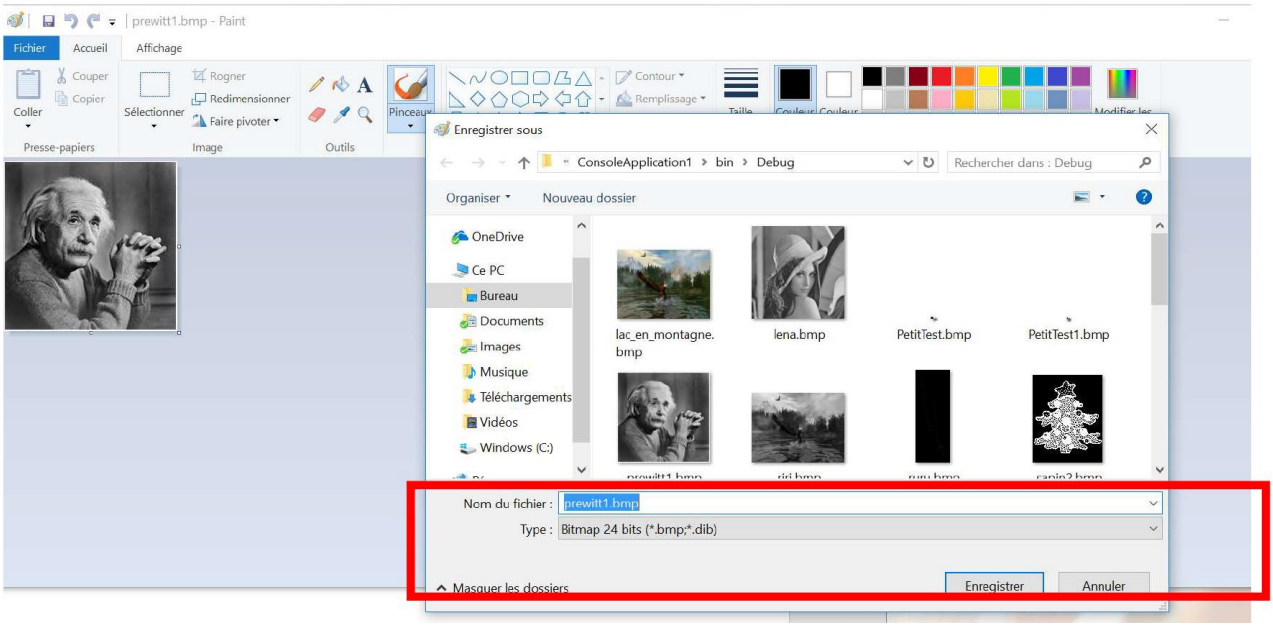
Initiation au traitement d'images en c#

Projet proposé par Mme Aline Ellul

L'objectif de cette réalisation est d'avoir une initiation au traitement d'images en c# à travers plusieurs séances de TD. Vous n'utiliserez que des images en format Bitmap 24bits.

L'idée est de créer un produit informatique qui lit une image dans un format donné (bitmap), traite cette image (agrandit, rétrécit …) et sauvegarde l'image dans un fichier de sortie différent de celui donné en entrée (toujours format Bitmap). Attention, il ne faut pas écraser les images d'origines.

Vous allez travailler sur des images de type Bitmap non compressé, 24 bits (un octet par couleur RGB). Dans un premier temps, l'idée est de comprendre la classe Bitmap fournie par c# puis dans un second temps de la réécrire en ajoutant des fonctionnalités.



On vous invite à consulter quelques liens pour vous familiariser avec le monde du traitement d'images et son vocabulaire : format, bitmap, pixel

https://en.wikipedia.org/wiki/Bitmap

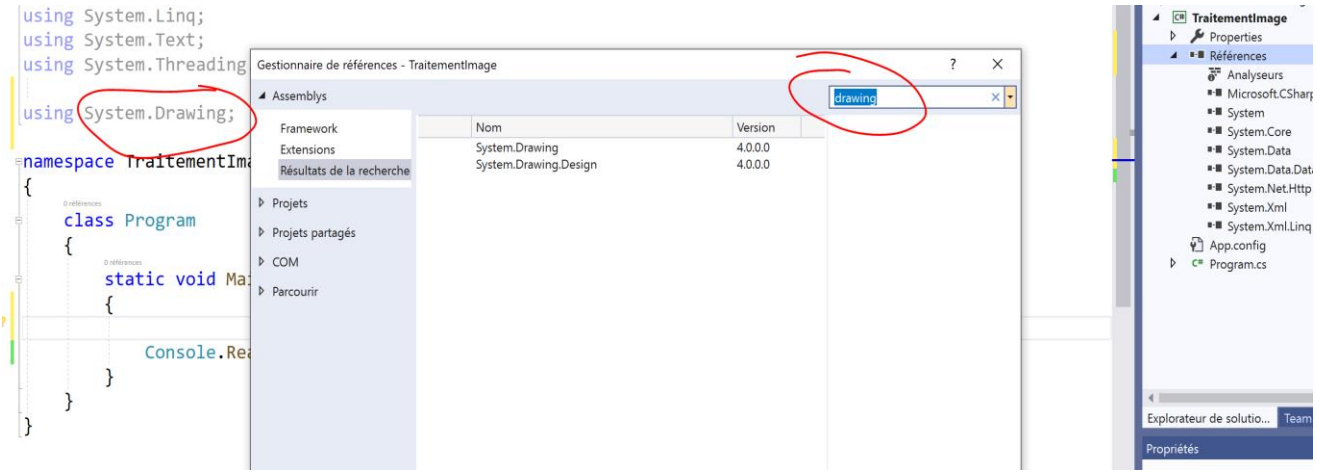
https://en.wikipedia.org/wiki/Pixel

https://fr.wikipedia.org/wiki/Rouge_vert_bleu

Partie 1 : Classe Bitmap du c#

Dans un premier temps, on va d'abord utiliser la classe Bitmap existante en c# pour mieux comprendre le fonctionnement et les fonctionnalités proposées.

Pour cela, vous devez tout d'abord créer un projet/solution sur Visual Studio puis vous insérez la bibliothèque using System.Drawing; dans votre fichier .cs et vous ajoutez les deux références Drawing comme ci-dessous




```
public sealed class Bitmap : Image
{
    public Bitmap(string filename);
    public Bitmap(Stream stream);
    public Bitmap(Image original);
    public Bitmap(string filename, bool useIcm);
    public Bitmap(Type type, string resource);
    public Bitmap(Stream stream, bool useIcm);
    public Bitmap(int width, int height);
    public Bitmap(Image original, Size newSize);
    public Bitmap(int width, int height, PixelFormat format);
    public Bitmap(int width, int height, Graphics g);
    public Bitmap(Image original, int width, int height);
    public Bitmap(int width, int height, int stride, PixelFormat format, IntPtr scan0);

    public static Bitmap FromHicon(IntPtr hicon);
    public static Bitmap FromResource(IntPtr hinstance, string bitmapName);
    public Bitmap Clone(RectangleF rect, PixelFormat format);
    public Bitmap Clone(Rectangle rect, PixelFormat format);
    public IntPtr GetHbitmap();
    public IntPtr GetHbitmap(Color background);
    public IntPtr GetHicon();
    public Color GetPixel(int x, int y);
    public BitmapData LockBits(Rectangle rect, ImageLockMode flags, PixelFormat format);
    public BitmapData LockBits(Rectangle rect, ImageLockMode flags, PixelFormat format, BitmapData bitm
    public void MakeTransparent();
    public void MakeTransparent(Color transparentColor);
    public void SetPixel(int x, int y, Color color);
    public void SetResolution(float xDpi, float yDpi);
    public void UnlockBits(BitmapData bitmapdata);
}
```

A partir de là, vous pourrez utiliser la classe Bitmap ainsi que toutes ses méthodes.

```
Bitmap bmpImage = new Bitmap("test3.bmp");
Console.WriteLine("width: " + bmpImage.Width + " height: " + bmpImage.Height);
for (int ii = 0; ii < bmpImage.Height; ii++)
    for (int j = 0; j < bmpImage.Width; j++)
    {
        Color c = bmpImage.GetPixel(j, ii);
        #region
        bmpImage.SetPixel(j, ii, Color.FromArgb(c.A, val, val, val));

    }
bmpImage.Save("testbitmap.bmp");
```

Sur le lien suivant, vous découvrirez la structure d'un fichier Bitmap

https://fr.wikipedia.org/wiki/Windows_bitmap

https://en.wikipedia.org/wiki/BMP_file_format

http://www.proftnj.com/RGB3.htm

Lorsque vous allez créer votre image de sortie, vous pouvez utiliser l'instruction suivante pour faciliter la visualisation du résultat (si perroquet.bmp est une image stockée sous le répertoire bin/debug de votre solution)

```
using System.IO;
using System.Diagnostics;

namespace LectureImage
{
    class Program
    {
        static void Main(string[] args)
        {
            Process.Start("perroquet.bmp");
            Console.ReadLine();
        }
    }
}
```

Partie 2 :

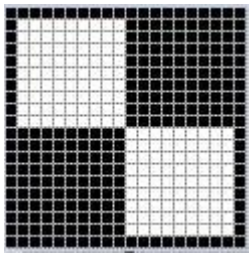
L'objectif de cette réalisation est d'avoir quelques opérations liées au traitement d'image sans utiliser la classe Bitmap. Vous aurez à faire :

1. Lire et écrire une image (à partir d'un format .bmp)
2. Traiter une image :
    - Agrandir/Rétrécir une image
    - Passage d'une photo couleur à une photo en nuances de gris et en noir et blanc
    - Superposition de 2 images
    - Miroir d'une image
    - Rotation d'un angle (90 ou 180). Bonus : rotation en fonction d'un angle quelconque (45°, 270°, 60° etc)

http://wxfrantz.free.fr/index.php?p=format-bmp

Un petit exemple

Soit l'image suivante Test.bmp (sur quelques pixels pour simplifier la compréhension), chaque carré est un pixel grossi au maximum. Il s'agit d'une image de 20 pixels sur 20 pixels



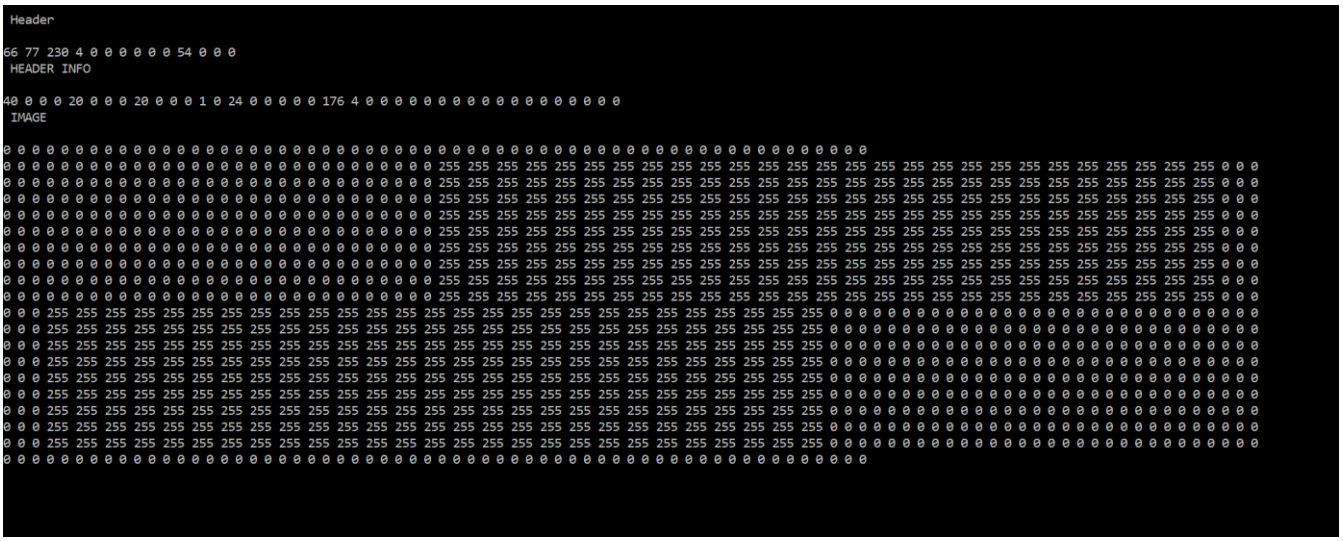
```
byte[] myfile = File.ReadAllBytes("./Images/Test.bmp");
//myfile est un vecteur composé d'octets représentant les métadonnées et les données de l'image

//Métadonnées du fichier
Console.WriteLine("\n Header \n");
for (int i = 0; i < 14; i++)
    Console.Write(myfile[i] + " ");
//Métadonnées de l'image
Console.WriteLine("\n HEADER INFO \n");
for (int i = 14; i< 54; i++)
    Console.Write(myfile[i] + " ");
//L'image elle-même
Console.WriteLine("\n IMAGE \n");
for (int i = 54; i < myfile.Length; i = i + 60)
{
    for (int j = i; j < i + 60; j++)
    {
        Console.Write(myfile[j] + " ");

    }
    Console.WriteLine();
}

File.WriteAllBytes("./Images/Sortie.bmp", myfile);
```

Le résultat de ce code affiche les informations suivantes :



Chaque ligne de l'image doit comporter un nombre total d'octets qui soit un multiple de 4; si ce n'est pas le cas, la ligne doit être complétée par des 0 de telle manière à respecter ce critère. Vous ne traiterez que des images multiples de 4.

Explication

Les 14 premiers octets décrivent le header de votre fichier (entête)

- 66 et 77 correspondent au code ascii des lettres B et M pour Bitmap
- 230 4 0 0 indique la taille du fichier en octets sur 4 octets à traduire en base décimale etc …

Vous noterez que toutes les tailles sont données dans le format « little endian »

https://fr.wikipedia.org/wiki/Endianness

Il faudra donc convertir ces tailles en entier en tenant compte de ce mode de formatage.
Les 40 autres octets suivants décrivent des informations liées à l'image.

- taille du header d'info (40 0 0 0) sur 4 octets
- largeur en pixels sur 4 octets (20 0 0 0)
- hauteur en pixel sur 4 octets (20 0 0 0)
- …
- Nombre d'octets par couleur (24 0) sur 2 octets
- ….

L'image démarre au 54ème octet. Le code RGB pour un pixel noir est égal à 0 0 0, le code RGB pour un pixel blanc est égal à 255 255 255

L'objectif de la première séance de TD est donc de lire une image et de convertir cette image (fichier binaire) en une instance de classe MyImage que vous devez définir. Cette classe doit contenir les informations générales sur l'image et l'image par elle-même. Vous ferez en sorte que les données ne soient jamais dupliquées (il est hors de question d'avoir comme attributs une matrice de bytes et une matrice de pixels, c'est-à-dire la même donnée sous 2 formats différents)
Nous retiendrons les informations suivantes :

- type d'image (BM par exemple),
- taille du fichier (int), taille Offset (int),
- largeur et hauteur de l'image (int)
- nombre de bits par couleur(int)
- l'image par elle-même sur laquelle vous ferez les traitements proposés ensuite. (matrice de RGB)

Vous pouvez concevoir éventuellement d'autres classes qui simplifieront la lisibilité et la sémantique du code.

Pour vous aider, vous créerez au moins pour la classe MyImage les constructeurs et méthodes suivantes :

- **public MyImage(string myfile)** lit un fichier (.bmp) et le transforme en instance de la classe Image
- **public void From_Image_To_File(string file)** prend une instance de MyImage et la transforme en fichier binaire respectant la structure du fichier .bmp
- **public int Convertir_Endian_To_Int(byte[] tab …)** convertit une séquence d'octet au format little endian en entier
- **public byte[] Convertir_Int_To_Endian(int val …)** convertit un entier en séquence d'octets au format little endian

Partie 3

- Appliquer un filtre (matrice de convolution) sur votre image
  - Détection de contour (https://docs.gimp.org/fr/plug-in-convmatrix.html)
  - Renforcement des bords (https://docs.gimp.org/fr/plug-in-convmatrix.html)
  - Flou (https://docs.gimp.org/fr/plug-in-convmatrix.html)
  - Repoussage (https://docs.gimp.org/fr/plug-in-convmatrix.html)

- Créer une image nouvelle (à partir de rien)
    - Créer une image décrivant une forme géométrique o Créer un histogramme se rapportant à une image https://fr.wikipedia.org/wiki/Histogramme_(imagerie_num%C3%A9rique)

➢ Innovation selon votre créativité ➔ sur cette section chacun devra se démarquer en proposant une réalisation personnelle et créative sur une ou plusieurs images.