INF1010

Programmation Orientée-Objet

Travail pratique #3
Héritage

Objectifs :	Permettre à l'étudiant de se familiariser avec l'héritage.
Remise du travail :	Lundi le 23 octobre 2017, 8h
Références :	Notes de cours sur Moodle & Chapitre 19 du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Informations préalables

La directive de précompilation « #ifndef »

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'en-tête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entraînerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H

#define NOMCLASSE_H

// Définir la classe NomClasse ici

#endif
```

La directive de précompilation « #include»

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

Travail à réaliser

Le travail consiste à continuer l'application de la gestion d'images commencée au TP 1 et 2 et d'y intégrer des notions d'héritage.

L'héritage permet de créer une classe de base possédant ses propres caractéristiques et de l'utiliser pour créer de nouvelles classes auxquelles on peut attribuer la caractéristique «est un». Par exemple, dans le cas du TP, un pixel de couleur est un pixel, ce qui donne lieu à une classe PixelCouleur qui dérive de la classe de base Pixel.

Dans ce TP, vous serez amené à lire les images fournies dans le dossier «Ensemble d'images» grâce à une fonction qui vous est déjà fournie dans la classe Image (fonction lireImage). Vous pourrez ensuite les modifier en utilisant les fonctions que vous allez créer en suivant les instructions données pour chaque classe et vous pourrez les enregistrer sur le disque et observer les changements avec la fonction sauvegarderImage.

Pour réussir ce TP il faudra vous familiariser avec la fonction C++ static_cast<T>(Objet). Cette fonction permet, en quelque sorte, d'informer le compilateur que l'objet auquel on fait affaire est d'un type différent de celui initialement attribué. Dans le cas du TP, elle servira à transformer un pointeur du type Pixel vers un pointeur du type réel de ce pixel (PixelCouleur, PixelGris, PixelNB)., voir les instructions qui suivent.

Le chemin d'un fichier représente sa position sur le disque dur. Dans le cas de ce TP il s'agit d'un string de la forme : /Users/NOMUSAGER/TP3/Ensemble d'images/Originale/RM.bmp. On peut aussi l'écrire sous la forme relative. Par exemple, si le travail se fait déjà dans le dossier TP3 de l'exemple précédent, alors un chemin relatif de fichier serait : ./Ensemble d'image/Originale/RM.bmp Le point au début du chemin du fichier indique le répertoire courant. Si vous avez des questions à ce sujet, n'hésitez pas à demander de l'aide aux chargés de TP.

Pour vous aider, les fichiers du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs ou méthodes qui ne sont plus nécessaires ont été supprimés. Et les méthodes à modifier vous ont été indiquées.

ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.

ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP2.

Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h

Classe *Pixel*

Cette classe caractérise un pixel simple. La classe Pixel précédemment utilisée se comparerait mieux à la classe PixelCouleur qui sera discutée plus tard dans ce TP. Pour ce TP, chaque Pixel aura un type dont les appellations sont fournies dans le fichier TypePixel.h. Vous devez absolument utiliser ces Types tout au long du TP. Le constructeur par défaut de cette classe donne le type TypePixel ::Couleur aux pixels.

Cette classe contient l'attribut privé suivant :

- **type_** : le type du pixel. Dois être un des TypePixel fournis dans TypePixel.h

Les méthodes suivantes doivent être implémentées :

- Le **constructeur par paramètres** qui construit un Pixel avec un **TypePixel**.
- **getType** : une fonction qui retourne le type du Pixel.
- **L'opérateur ==** qui prend un *Pixel* en paramètre et permet de comparer 2 pixels. Cet opérateur va pouvoir faire l'opération *pixel1==pixel2*.
- Un **destructeur**.

Classe *PixelCouleur*

La classe PixelCouleur ressemble beaucoup à la classe Pixel des TP 1 et 2. Cette classe permet d'enregistrer les valeurs RGB d'un pixel d'une image dans un tableau de unsigned char. Un unsigned char n'est qu'un emplacement mémoire de 8 bits qui ne peut donc que prendre des valeurs comprises dans l'intervalle [0, 255]. Cette classe possède une énumération R, G et B qui servent à définir la position des valeurs rouges (R), vertes (G) et bleue (B) dans le tableau de unsigned char. Pour accéder à ce tableau, utilisez l'énumération Couleur::R, Couleur ::G et Couleur::B qui ont respectivement les valeurs 0, 1 et 2.

Cette classe contient l'attribut privé suivant :

- **donnee_** : un tableau de unchar (unsigned char) d'une grandeur maximale de 3 dont la première valeur et le R, la seconde le G et la dernière le B.

Les méthodes suivantes doivent être implémentées :

- Un **constructeur par défaut** qui initialise donnee_ a un tableau de 3 unchar dont les valeurs sont R=0, G=0 et B=0.
- Un **constructeur par paramètre** acceptant des unchar r, g et b qui serviront à remplir les valeurs de l'attribut donnee_.
- Méthode **convertirPixelBN()** qui retourne la valeur booléenne true si la moyenne des valeurs RGB est supérieure à 127 et false si elle est égale ou inférieure à 127.
- Méthode **convertirPixelGris()** qui retourne la moyenne des valeurs RGB.
- Méthode **retournerR()** qui retourne la valeur de la teinte rouge (R).

- Méthode **retournerG()** qui retourne la valeur de la teinte verte (G).
- Méthode **retourherB()** qui retourner la valeur de la teinte bleue (B).
- L'**opérateur ==** qui permet de comparer deux PixelCouleur. Il doit comparer leur type ainsi que les valeurs RGB.
- Un **destructeur**.

Classe *PixelGris*

La classe PixelGris ressemble énormément à la classe PixelCouleur, mais ne contient qu'une simple valeur unchar comme attribut. Un équivalent serait un PixelCouleur dont les valeurs RGB seraient toutes les mêmes.

Cette classe contient l'attribut privé suivant :

- **donnee_** : un unchar.

Les méthodes suivantes doivent être implémentées :

- Un **constructeur par défaut** qui initialise donnee_ à 0.
- Un **constructeur par paramètre** acceptant un unsigned int qui ira dans donnee_.
- Méthode **convertirPixelBN()** qui retourne la valeur booléenne true si donnee_ a une valeur supérieure à 127 et false si sa valeur est égale ou inférieure à 127.
- Méthode **convertirPixelCouleur()** qui retourne un pointeur unchar* sur un tableau comportant les valeurs RGB équivalentes donnant un PixelCouleur.
- Méthode **obtenirDonnee()** qui retourne la valeur contenue dans donnee_.
- L'**opérateur ==** qui permet de comparer deux PixelGris. Il doit comparer leur type ainsi que la valeur de donnee_.
- Un **destructeur**.

Classe *PixelBN*

La classe PixelBN ne possède qu'un attribut booléen. Sa valeur indique que le pixel est soit noir, si sa valeur est false, soit blanc, si sa valeur est true. Il est donc comparable à un PixelCouleur dont les valeurs RGB seraient respectivement R=0, G=0, B=0 ou R-255, G=255, B=255.

Cette classe contient l'attribut privé suivant :

- **donnee_** : un booléen.

Les méthodes suivantes doivent être implémentées :

- Un **constructeur par défaut** qui initialise donnee_ à false.
- Un **constructeur par paramètre** acceptant un bool qui ira dans donnee_.

- Méthode **convertirPixelGris()** qui retourne un unchar de la valeur équivalente de donnee_ pour un PixelGris().
- Méthode **convertir PixelCouleur()** qui retourne un unchar* pointant sur un tableau des valeurs équivalentes RGB qui permettrait de construire un PixelCouleur.
- Méthode **obtenirDonnee()** qui retourne la valeur contenue dans donnee_.
- L'**opérateur ==** qui permet de comparer deux PixelBN. Il doit comparer leur type ainsi que la valeur donnee_.
- Un **destructeur**.

Classe Image

Cette classe caractérise une image par les attributs cheminVersImageOriginale_, nomDuFichier_, hauteur_, largeur_, taille_ et typeImage. Elle possède deux méthodes déjà implémentées pour vous, soit la méthode lireImage qui permet de lire une image à partir du disque dur et la méthode sauvegarderImage qui permet de sauvegarder une image. **ATTENTION**, le tableau de pixels n'est pas en 2 dimensions. Il s'agit d'un tableau dont la taille est égale à la multiplication entre la hauteur de l'image et la largeur de l'image, voir figure 1.

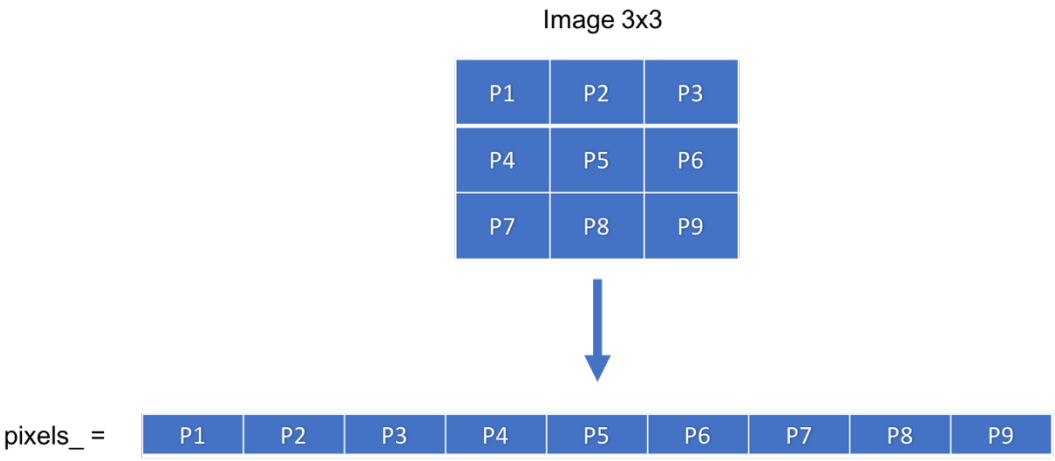


Figure 1 : Représentation d'une image après son stockage dans pixels_

Vous n'aurez pas besoin de cette information, mais cela signifie que l'indice d'un pixel (x, y) peut être calculé par la formule

```
Unsigned int indice = y * largeur_ + x;
```

Exemple, l'indice du pixel P1 dans pixels_ serait 0, car sa position est (0, 0) dans l'image. Alors que l'indice du pixels P6 serait 5, car sa position est (1,2).

À l'inverse, il est possible de retrouver le x par la formule :

```
Unsigned int x = indice / 3; (division entire donc aucun reste)
```

Et le y par la formule :

```
Unsigned int y = indice % 3; (% veut dire modulo)
```

La fonction static_cast<>() sera nécessaire pour la réalisation de cette classe.

Ne pas vous souciez de la dernière ligne du fichier Image.h:

```
size_t bitmap_encode_rgb(const unchar *rgb, unsigned int width, unsigned int
height, unsigned char** output);
```

Il s'agit d'une fonction utilisée pour sauvegarder l'image sur le disque.

Cette classe contient l'attribut privé suivant :

- **cheminVersImageOriginale_** : un string donnant le chemin jusqu'à l'image originale permettant de créer l'image
- **nomDuFichier_** : un string contenant le nom de l'image obtenue à partir du chemin de fichier.
- **pixels_** : un tableau des pixels associés à l'image. **Attention, ce tableau n'est pas comme celui construit précédemment dans les autres TP, voir figure 1 et se rempli tout seul lors de l'appel à lireImage!!**
- **hauteur_** : la hauteur de l'image en pixels **s'initialise lors de l'appel à lireImage**.
- **largeur_** : la largeur de l'image en pixels **s'initialise lors de l'appel à lireImage**.
- **taille_** : la taille de l'image en pixels (hauteur_ * largeur_) **s'initialise lors de l'appel à lireImage.**
- **typeImage_** : le type de l'image suivant le header TypeImage.h

Les méthodes suivantes doivent être implémentées :

- Un **constructeur par défaut** qui initialise tout à 0 et nullptr sauf le TypeImage_ qui sera à Couleurs.
- Un **constructeur par copie** (si nécessaire) qui doit copier les informations en accordance avec le nouveau tableau pixels_.
- Un **constructeur par paramètre** qui initialise TypeImage_ au type donné en paramètre, cheminVersImageOriginale au cheminDuFichier, le nomDuFichier à partir du cheminDuFichier et qui appelle lireImage sur le cheminVersImageOriginale et avec le TypeImage en paramètre.
- **obtenirCheminDuFichier** qui retourne le cheminDuFichier.
- **obtenirNomImage** qui retourne le nom de l'image.
- **obtenirTaille** qui retourne la taille.
- **obtenirType** qui retourne le type.
- **obtenirTypeEnString** qui retourne le type de l'image sous la forme d'un string. (Fournit)
- **changerNomImage** qui change le nom de l'image

- **L'opérateur =** qui écrase les attributs de l'objet de gauche par les attributs de l'objet passé en paramètre. (attention aux fuites de mémoires) à modifier pour qu'il fonctionne avec la nouvelle représentation de l'image dans pixels_, voir figure 1.
- L'**opérateur <<** qui affiche les informations d'une image en **suivant la forme donnée à la fin du document**.
- L'**opérateur ==** qui prend une image en paramètre et qui permet de comparer 2 images en considérant le nom et l'ensemble des pixels. Cet opérateur va pouvoir effectuer l'opération *image1==image2*, à modifier pour qu'elle fonctionne avec la nouvelle représentation de l'image dans pixels_, voir figure 1.
- L'**opérateur ==** qui prend un nom en paramètre et compare une image avec une chaîne de caractère. L'opérateur retourne *true* si les noms sont identiques et false sinon. Ainsi, cet opérateur va pouvoir faire l'opération *image==nom*.
- L'**opérateur ==** de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *nom==image*.
- **convertirNB** qui convertit tous les pixels de l'image en PixelNB, sauf si l'image est déjà en noir et blanc (TypeImage ::NB). (static_cast devra être utilisé)
- **convertirGris** qui convertit tous les pixels de l'image en PixelGris, sauf si l'image est déjà en gris (TypeImage ::Gris). (static_cast devra être utilisé)
- **convertirCouleur** qui convertit tous les pixels de l'image en PixelCouleru, sauf si l'image est déjà en couleur (TypeImage ::Couleurs). (static_cast devra être utilisé)
- Un **destructeur**.

Classe *GroupImage*

Cette classe regroupe un ensemble d'images.

Les attributs et méthodes liées au TP précédent restent inchangés, sauf si spécifié

Les méthodes suivantes doivent être implémentées :

- L'**opérateur <<** affiche les informations qui concerne un groupe d'images suivant l'exemple présenté à la fin du document.
- **toutMettreEnNB** qui convertit toutes les images en NB.
- **toutMettreEnGris** qui convertit toutes les images en Gris.
- **toutMettreEnCouleur** qui convertit toutes les images en couleur.
- **toutEnregistrer** qui enregistre toutes les images du groupe en s'assurant d'envoyer les images Couleurs dans le sous dossier Couleurs du dossier Ensemble d'image, les images grises dans le sous dossier Nuances de Gris du dossier Ensemble d'images et les images en noire et blanc dans le sous dossier Noir et Blanc du dossier Ensemble d'images. Indice, la fonction sauvegarderImage de la classe image sauvegarde les images à la destination fournie et la fonction obtenirTypeEnString retourne les strings Couleurs, Nuances de Gris et Noir et Blanc selon le type de l'image correspondant.

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées. Votre affichage devrait avoir une apparence semblable à celle ci-dessous. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable ainsi que de choisir vos propres exemples de noms, prénoms, spécialité, etc. :

Spécifications générales

• Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent
• Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs
• Ajouter le mot-clé const chaque fois que cela est pertinent
• Appliquez un l'affichage présenté à la fin de ce document
• Documenter votre code source
• Note : Lorsqu'il est fait référence aux valeurs par défaut, pour un string cela équivaut à la chaîne vide, et pour un entier au nombre 0

Questions

1. Pourquoi est-il logique de dériver une classe PixelCouleur d'une classe Pixel?

2. Considérez la situation suivante :

```
Pixel p1 = PixelCouleur(20, 30, 50);
Pixel p2 = PixelCouleur(12, 140, 86);

bool b = ( p1 == p2 );
cout << b << end;
```

Est-il possible d'assigner à p1 et p2 des PixelCouleur? Pourquoi?
Quelle sera la valeur de b et pourquoi?

3. À quoi sert le static_cast?

Correction

La correction du TP se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (4 points) Comportement exact des méthodes du programme ;
- (3 points) Utilisation adéquate de l'héritage ;
- (2 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriées mémoire
- (3 points) Réponse aux questions.

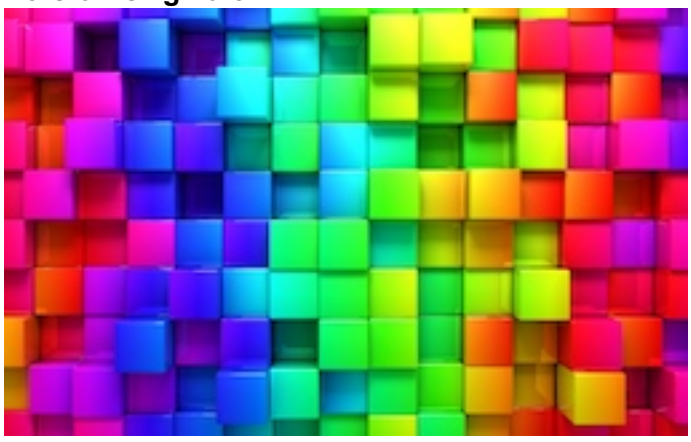
```
**************************************************
Creation du groupe d'image
**************************************************

Breaking-Bad.bmp a bien ete ajoute
Couleur.bmp a bien ete ajoute
Fall.bmp a bien ete ajoute
RM.bmp a bien ete ajoute
SolarEclipse.bmp a bien ete ajoute
WiC.bmp a bien ete ajoute

**************************************************
Affichage des images du groupe :
**************************************************

Nom de l'image :  Breaking-Bad.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/Breaking-Bad.bmp
Taille de l'image : 65024 pixels
Resolution en pixels : 256 x 254
Type de l'image : Couleurs
--------------------------------------------------
Nom de l'image :  Couleur.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/Couleur.bmp
Taille de l'image : 40960 pixels
Resolution en pixels : 256 x 160
Type de l'image : Couleurs
--------------------------------------------------
Nom de l'image :  Fall.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/Fall.bmp
Taille de l'image : 40960 pixels
```

```
Nom de l'image :  Fall.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/Fall.bmp
Taille de l'image : 40960 pixels
Resolution en pixels : 256 x 160
Type de l'image : Couleurs
--------------------------------------------------
Nom de l'image :  RM.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/RM.bmp
Taille de l'image : 65536 pixels
Resolution en pixels : 256 x 256
Type de l'image : Couleurs
--------------------------------------------------
Nom de l'image :  SolarEclipse.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/SolarEclipse.bmp
Taille de l'image : 51968 pixels
Resolution en pixels : 256 x 203
Type de l'image : Couleurs
--------------------------------------------------
Nom de l'image :  WiC.bmp
Chemin de l'original : ./Ensemble d'images/Originale/Low Res/WiC.bmp
Taille de l'image : 40960 pixels
Resolution en pixels : 256 x 160
Type de l'image : Couleurs
--------------------------------------------------


L'image Breaking-Bad.bmp est deja en couleur, conversion innutile.
L'image Couleur.bmp est deja en couleur, conversion innutile.
L'image Fall.bmp est deja en couleur, conversion innutile.
L'image RM.bmp est deja en couleur, conversion innutile.
```

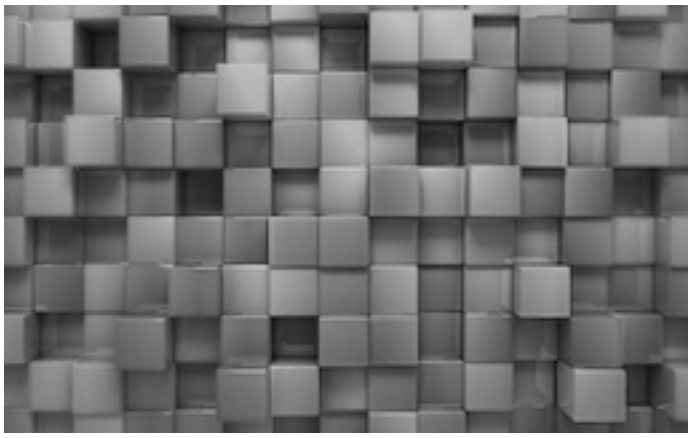
```
Sauvegarde de ./Ensemble d'images/Couleurs/im0Breaking-Bad.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im1Couleur.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im2Fall.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im3RM.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im4SolarEclipse.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im5WiC.bmp

Conversion de l'image Breaking-Bad.bmp
Conversion de l'image Couleur.bmp
Conversion de l'image Fall.bmp
Conversion de l'image RM.bmp
Conversion de l'image SolarEclipse.bmp
Conversion de l'image WiC.bmp

Sauvegarde de ./Ensemble d'images/Nuances de Gris/im0Breaking-Bad.bmp
Sauvegarde de ./Ensemble d'images/Nuances de Gris/im1Couleur.bmp
Sauvegarde de ./Ensemble d'images/Nuances de Gris/im2Fall.bmp
Sauvegarde de ./Ensemble d'images/Nuances de Gris/im3RM.bmp
Sauvegarde de ./Ensemble d'images/Nuances de Gris/im4SolarEclipse.bmp
Sauvegarde de ./Ensemble d'images/Nuances de Gris/im5WiC.bmp

Conversion de l'image Breaking-Bad.bmp
Conversion de l'image Couleur.bmp
Conversion de l'image Fall.bmp
Conversion de l'image RM.bmp
Conversion de l'image SolarEclipse.bmp
Conversion de l'image WiC.bmp

Sauvegarde de ./Ensemble d'images/Noir et Blanc/im0Breaking-Bad.bmp
Sauvegarde de ./Ensemble d'images/Noir et Blanc/im1Couleur.bmp
```

Exemple de transformation d'image :

Version Originale



Version Nuances de Gris :



Version Noir et Blanc :