INF1010

Programmation Orientée-Objet

Travail pratique #5
Classes génériques et la STL

Objectifs :	Permettre à l'étudiant de se familiariser avec le concept de fonctions et des classes générique ainsi qu'introduction à la librairie STL.
Remise du travail :	Lundi le 20 novembre 2017, 8h
Références :	Notes de cours sur Moodle & Chapitres 11 à 14, 16 et 20 du livre Big C++ 2e éd.
Documents à remettre :	Les fichiers .cpp et .h complétés ainsi que le répertoire d'images 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 effectué dans ce TP continue celui amorcé par les TP1, 2,3 et 4, soit une application de gestion d'images en y intégrant les notions de foncteurs et utiliser les structures de donnés disponibles avec la librairie STL.

Les foncteurs sont des objets qui peuvent agir comme des fonctions grâce à la surcharge de l'opérateur « () ». Les foncteurs sont très utiles lorsqu'on travaille avec des structures de donnés provenant de la librairie STL comme list ou queue. Les foncteurs peuvent avoir aucun ou plusieurs attributs. Généralement, un foncteur sans attribut permet de manipuler les objets sur lesquels il est appliqué. Les foncteurs à un attribut sont souvent utilisés à des fins de comparaison, mais aussi d'autres utilités.

Tout comme pour le TP3, 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 les concepts de foncteurs ainsi que plusieurs structures de donnés de la STL. La documentation complète de la STL est disponible en ligne et un aide-mémoire est présent sur Moodle. L'utilisation du **static_cast** est **INTERDITE**. Tout le TP est réalisable sans cette fonction.

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 TP4.

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

Classe *Pixel*

Cette classe caractérise une interface pixel simple. L'attribut **type_** devra être **supprimé**. Et des fonctions virtuelles et virtuelles pures devront être définies.

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

Méthodes à ajouter :

- **estMajoriteRouge**: Fonction virtuelle pure retournant un bool si la couleur rouge est dominante (A>B && A >C ou A est la couleur « majoritaire »)
- **estMajoriteVert**: Fonction virtuelle pure retournant un bool si la couleur verte est dominante (A>B && A > C ou A est la couleur « majoritaire »).
- **estMajoriteBleu**: Fonction virtuelle pure retournant un bool si la couleur bleue est dominante (A>B && A > C ou A est la couleur « majoritaire »).
- **retournerIntensiteMoyenne** : Fonction virtuelle pure qui retourne double d'l'intensité moyenne des trois couleurs normalisée (entre 0 et 1).

Classe *PixelCouleur*

Cette classe dérive de Pixel et 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.

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

Les méthodes suivantes doivent être ajoutées :

- **estMajoriteRouge**: Fonction virtuelle retournant un bool si la couleur rouge est (A>B && A > C ou A est la couleur « majoritaire »).
- **estMajoriteVert**: Fonction virtuelle retournant un bool si la couleur verte est dominante (A>B && A > C ou A est la couleur « majoritaire »).
- **estMajoriteBleu**: Fonction virtuelle retournant un bool si la couleur bleue est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **retournerIntensiteMoyenne** : Fonction virtuelle qui retourne un double d'l'intensité moyenne des trois couleurs normalisée (entre 0 et 1). L'intensité d'une couleur est sa valeur entre 0 et 255.

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 dérive de l'interface Pixel.

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

Les méthodes suivantes doivent être ajoutées :

- **estMajoriteRouge**: Fonction virtuelle retournant un bool si la couleur rouge est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **estMajoriteVert**: Fonction virtuelle retournant un bool si la couleur verte est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **estMajoriteBleu**: Fonction virtuelle retournant un bool si la couleur bleue est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **retournerIntensiteMoyenne** : Fonction virtuelle qui retourne un double d'l'intensité moyenne du taux de gris normalisé (entre 0 et 1). L'intensité d'une couleur est sa valeur entre 0 et 255.

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 dérive de l'interface Pixel.

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

Les méthodes suivantes doivent être ajoutées :

- **estMajoriteRouge**: Fonction virtuelle retournant un bool si la couleur rouge est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **estMajoriteVert**: Fonction virtuelle retournant un bool si la couleur verte est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **estMajoriteBleu**: Fonction virtuelle retournant un bool si la couleur bleue est dominante (A>B && A > C ou A est la couleur « majoritaire »)
- **retournerIntensiteMoyenne** : Fonction virtuelle qui retourne un double d'l'intensité moyenne normalisé (entre 0 et 1). L'intensité d'une couleur est sa valeur entre 0 et 255.

-

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.

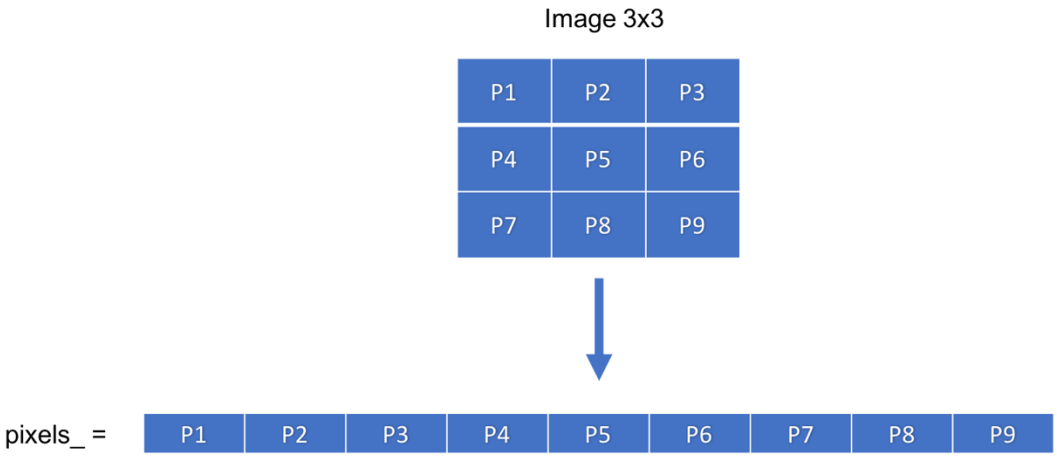


Figure 1 : Représentation d'une image après son stockage dans pixels_

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 pixel 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 entière donc aucun reste)
```

Et le y par la formule :

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

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.

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

Les méthodes suivantes doivent être ajoutées :

- **estMajoriteRouge**: Fonction retournant un bool si la couleur rouge est dominante majoritaire (50% + 1 pixels) dans l'image.
- **estMajoriteVert**: Fonction retournant un bool si la couleur verte est dominante majoritaire (50% + 1 pixels) dans l'image.
- **estMajoriteBleu**: Fonction retournant un bool si la couleur bleue est dominante majoritaire (50% + 1 pixels) dans l'image.
- **obtenirIntensiteMoyenne**: Fonction qui retourne un double d'l'intensité moyenne normalisé (entre 0 et 1) dans la totalité des pixels. L'intensité d'une couleur est sa valeur entre 0 et 255.

Remarque : Les 4 méthodes doivent utiliser les méthodes virtuelles ajoutés dans Pixel et ses classes dérivées.

Classe *GroupImage*

Cette classe regroupe un ensemble d'images.

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

Les attributs à modifier :

L'attribut « images_ » doit être transformé d'un **vector** en **list**. Cette modification implique la modification d'un certain nombre de méthodes de la classe GroupImage. **Vous devez changer tous les méthodes affectées par ce changement.**

Remarque : Les modifications doivent utiliser les méthodes de la classe list de la STL ainsi que les itérateurs de list. Vous avez le droit d'utiliser le mot clé « auto » pour simplifier votre code.

ATTENTION : L'utilisation de boucles for ou while de la forme for(int i; i <vec.size(); i++) est interdit pour les nouvelles méthodes que vous devez implémenter. Vous devez soit utiliser les algorithmes lorsque possible, soit utiliser les boucles for/while en utilisant les itérateurs.

Les méthodes suivantes doivent être modifiées :

- **operator=** : Utiliser la fonction **insert** et les itérateurs pour copier les images.
- **ajouterImage et retirerImage :** Les boucles for() classiques sont interdites, utiliser les algorithmes de recherche.
- **obtenirImage :** la fonction doit prendre en paramètre un `const std::string& nom`

  et les boucles for() classiques sont interdites, utiliser les algorithmes de recherche.

Les méthodes suivantes doivent être ajoutées :

- **obtenirIntensiteMoyenne** : Fonction retournant un double représentant l'intensité moyenne de tous les images dans le groupe.
- **obtenirTailleMoyenne :** Fonction retournant un double représentant la taille moyenne de tous les images dans le groupe.
- **appliquerFoncteurUnaire :** Fonction générique qui prend un prédicat en paramètre. La fonction applique le foncteur à toutes les images du conteneur, notez que l'application du foncteur sur un élément peut modifier le foncteur.
- **supprimerElements :** Fonction générique qui prend un prédicat unaire en paramètre. La fonction supprime tous les éléments du conteneur pour lesquels l'application du foncteur passé en argument retourne true. Pensez à utiliser la méthode remove_if appropriée (celle de la STL? Celle du conteneur ?)

BaseDeDonnesGenerique.h

Cette classe représente une base de donnés générique avec 2 paramètres <T,S>.

Remarque : Toutes les fonctions doivent être implémentés dans ce fichier

Attributs de la classe :

- list<T*> listImage_ : contient des Image.
- list<S*> listGroupeImage_ : contient des GroupeImage

Les méthodes à implémenter :

- **obtenirListImages** : retourne la liste des Images.
- **obtenirListGroupeImages :** retourne la list des GroupeImage.
- **ajouter(T*)** : ajoute un objet de type T dans la listImage_
- **ajouter(S*) :** ajoute un objet de type S dans la listGroupeImage_
- **supprimer(T*)** : supprime un objet de type T de listImage_. Vous devez utiliser les itérateurs.
- **supprimer(S*)** : supprime un objet de type S de listGroupeImage_. Vous devez utiliser les itérateurs.
- **supprimerImage() :** La fonction supprime tous les éléments du conteneur listeImage_ pour lesquels l'application du foncteur passé en argument retourne true. Vous devez utiliser la fonction **remove_if** de la STL.
- **supprimerGroupeImage() :** La fonction supprime tous les éléments du conteneur listGroupImage_ pour lesquels l'application du foncteur passé en argument retourne true. Vous devez utiliser la fonction **remove_if** de la STL.
- **vider() :** vide les 2 listes

BaseDeDonnesSpecifique.cpp

Cette classe hérite de la classe BaseDeDonnesGenerique et concrétiise les types T et S et Image et GroupeImage. Cette classe a seulement 2 méthodes

Les méthodes à ajouter :

- **AssocierImage** : cette fonction prend en paramètre une image et un groupe d'images et rajoute l'image au groupe. Remarque : cette modification doit être reflétée dans le contenu de la base de données aussi.
- **EnleverImage :** cette fonction prend en paramètre un groupe d'images et un nom d'image et essaie d'enlever l'image du groupe. **Remarque : cette modification doit être reflétée dans le contenu de la base de données aussi.**
- **Operateur << :** cette méthode affiche le contenu de la base de données. Le format est laissé à votre discrétion, mais on voudrait minimalement voir le nombre de groupes et d'images dans la base de données ainsi que leur noms.

Foncteur.h

Ce fichier comporte les différents foncteurs utilisés dans le TP. Les foncteurs doivent être tous implémentés seulement dans ce fichier.

FoncteurGenerateurNombresAlea : ce foncteur génère des nombres aléatoire entre min et max ([min, max[ ). Le foncteur possède 2 attributs : min_ et max_ de type **unsigned int**;

FoncteurEgalImages : ce foncteur compare deux images et retourne **VRAI** si les deux images sont égales. Le foncteur possède 1 attribut : image_ de type **Image***.

FoncteurImagesDeMemeNom : ce foncteur compare deux images en fonction de leurs noms. Le foncteur possède 1 attribut : nom_ de type **string;**

FoncteurObtenirTailleImage : ce foncteur retourne la taille d'une image. Le foncteur n'a pas d'attributs.

FoncteurMettreEnGris : ce foncteur convertir une image en image grise. Le foncteur n'a pas d'attributs.

FoncteurMettreEnBN : ce foncteur convertir une image en image blanc-noir. Le foncteur n'a pas d'attributs.

FoncteurMettreEnCouleur : ce foncteur convertir une image en image colorée. Le foncteur n'a pas d'attributs.

FoncteurMettreEnNegatif : ce foncteur convertir une image en image négative. Le foncteur n'a pas d'attributs.

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.

Étant donné que vous n'avez pas vu toute la matière nécessaire, on vous recommande FORTEMENT de commencer par l'implémentation des classes de bases de données et les modifications des classes Pixel, PIxelCouleur, PixelGris et PixelBN pour la première semaine. Les méthodes qui nécessitent la STL dans BaseDeDonnesGenerique peuvent être implémentés plus tard sans que vous soyez bloqués dans les instructions de **main.cpp** pour la première partie.

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 affichage similaire à ce qui est 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. Quel est l'avantage d'utiliser une liste au lieu d'un vecteur si on veut supprimer ou rajouter un élément ? Quelle structure de données est plus avantageuse si on veut accéder à des éléments à des positions aléatoires ?

2. Pourquoi est-ce que l'implémentation des classes génériques est dans .h et pas séparée en .h et .cpp comme les classes normales ?

Correction

La correction du TP se fera sur 20 points.

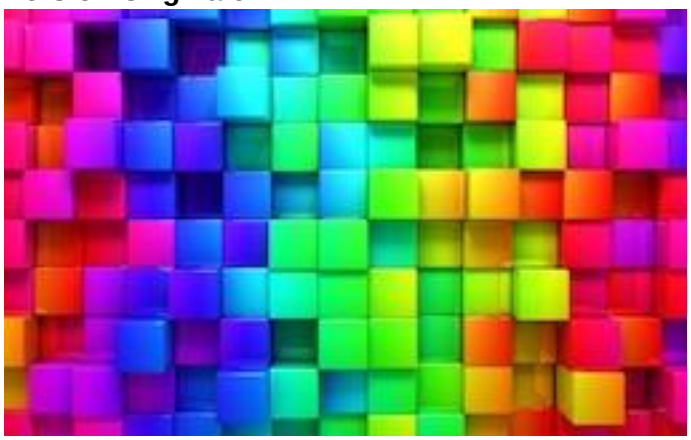
Voici les détails de la correction :

- (4 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (5 points) Comportement exact des méthodes du programme et l'utilisation de la STL;
- (4 points) Utilisation adéquate des foncteurs;
- (1 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriée mémoire
- (1 point) Réponse aux questions.

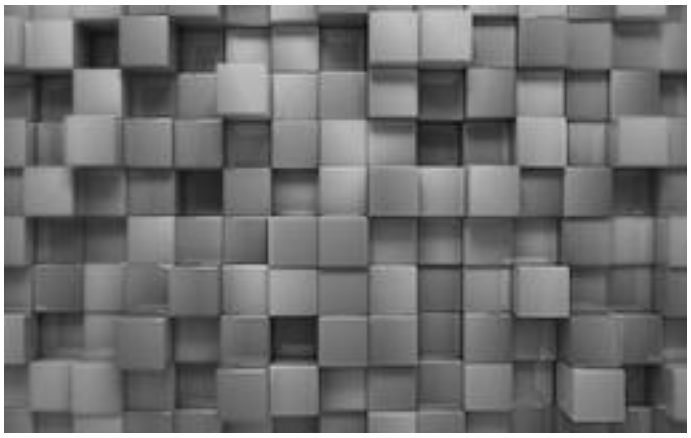
```
************************************************
Creation de la file de travail
************************************************

************************************************
CHARGEMENT DES IMAGES

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
Monkey.bmp a bien ete ajoute
WiC.bmp a bien ete ajoute
Sloth.bmp a bien ete ajoute
Green.bmp a bien ete ajoute
************************************************

************************************************
CALCUL DE LA MOYENNE

Moyenne du groupe 1 est de :0.506662

MOYENNES DES IMAGES DANS LA BASE DE DONNES:
Couleur.bmp avec moyenne de : 0.431773
Fall.bmp avec moyenne de : 0.326076
SolarEclipse.bmp avec moyenne de : 0.168303
Monkey.bmp avec moyenne de : 0.44783
WiC.bmp avec moyenne de : 0.241685
Green.bmp avec moyenne de : 0.390853
************************************************

************************************************
CALCUL DE LA TAILLE

Taille moyenne : 52821.3
Nombre d'images totales dans la base de donnes 9
Nombre d'images dont la taille est plus grande que la moyenne 5
************************************************

************************************************
CONVERSION EN NOIR ET BLANC

Sauvegarde de ./Ensemble d'images/Noir et Blanc/im0Fall.bmp
Sauvegarde de ./Ensemble d'images/Noir et Blanc/im1Monkey.bmp
Sauvegarde de ./Ensemble d'images/Noir et Blanc/im2Green.bmp
************************************************
```

Exemple de transformation d’image :

Version Originale



Version Nuances de Gris :



Version Noir et Blanc :

Version en négatif :