INF1010

Programmation Orientée-Objet

Travail pratique #4
Fonction virtuelle et Polymorphisme

Objectifs :	Permettre à l'étudiant de se familiariser avec les fonctions virtuelles et le concept de polymorphisme.
Remise du travail :	Lundi le 30 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 effectué dans ce TP continue celui amorcé par les TP1, 2 et 3, soit une application de gestion d'images en y intégrant les notions de fonctions virtuelles.

Les fonctions virtuelles permettent à une classe fille de surcharger une méthode et que celle-ci soit appelée par le compilateur C++ sur un pointeur de cette classe, même si le pointeur a été défini comme un pointeur de la classe mère. De plus, elles peuvent servir à définir des fonctions virtuelles pures en terminant la définition d'une fonction virtuelle par un = 0. Une classe possédant au moins une fonction virtuelle pure ne peut être instanciée et est appelée une interface. Ce sera le cas de notre classe Pixel pour le TP4.

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 fonction virtuelle et virtuelle pure, ainsi que sur les interfaces C++. 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 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 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

À modifier dans les attributs :

- **Supprimer l'attribut type_** : le type du pixel.

Méthodes à supprimer :

- **Supprimez** le **constructeur par paramètre**.
- **Supprimez** la méthode **getType.**

Méthodes à modifier :

- Le **constructeur par défaut** n'initialise plus rien.
- Rendre le **destructeur virtuel**
- **operateur==** : Doit devenir une fonction virtuelle qui vérifie que les valeurs R, G et B d'un pixel sont bien les mêmes. Doit être implémenté dans la classe Pixel.

Méthodes à ajouter :

- **retournerR** : Fonction virtuelle pure retournant un unchar correspondant à la teinte rouge.
- **retournerG** : Fonction virtuelle pure retournant un unchar correspondant à la teinte verte.
- **retournerB** : Fonction virtuelle pure retournant un unchar correspondant à la teinte bleue.
- **mettreEnNegatif** : Fonction virtuelle pure qui met le pixel en négatif.
- **retournerCopieProfonde** : Fonction virtuelle pure qui retourne un pointeur Pixel*.
- **convertirPixelBN** : Fonction virtuelle pure qui retourne la valeur du pixel équivalent s'il s'agissait d'un Pixel en Noir et Blanc sous la forme d'un bool.
- **convertirPixelGris** : Fonction virtuelle pure qui retourne la valeur du Pixel équivalent s'il s'agissait d'un Pixel en Nuance de Gris sous la forme d'un unchar.

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 supprimées :

Opérateur ==.

Les méthodes suivantes doivent être modifiées :

- Le **constructeur par défaut** (si nécessaire)
- Un **constructeur par paramètre** (si nécessaire)

Les méthodes suivantes doivent être ajoutées :

- **retournerCopieProfonde** : Fonction virtuelle qui retourne une copie profonde du pixel.
- **mettreEnNegatif** : Fonction virtuelle qui modifie les valeurs R, G et B en faisant 255 – la valeur courante.

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 supprimées :

Opérateur ==.

Les méthodes suivantes doivent être modifiées :

- Le **constructeur par défaut** (si nécessaire).
- Le **constructeur par paramètre** (si nécessaire).

Les méthodes suivantes doivent être ajoutées :

- **mettreEnNegatif :** Fonction virtuelle qui modifie l'attribut donnee_ par 255 – donnee_.
- **retournerR :** Fonction virtuelle qui retourne la valeur de R associée au pixel gris.
- **retournerG :**Fonction virtuelle qui retourne la valeur de G associée au pixel gris.
- **retournerB :** Fonction virtuelle qui retourne la valeur de B associée au pixel gris.
- **retournerUneCopieProfonde :** Fonction virtuelle qui retourne une copie profonde du pixel.

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 supprimées :

Opérateur ==.

Les méthodes suivantes doivent être modifiées :

- Le **constructeur par défaut** (si nécessaire).
- Le **constructeur par paramètre** (si nécessaire).

Les méthodes suivantes doivent être ajoutées :

- **mettreEnNegatif :** Fonction virtuelle qui modifie donnee_ par son inverse.
- **retournerR :** Fonction virtuelle qui retourne la valeur de R associée au pixel noir et blanc.
- **retournerG :**Fonction virtuelle qui retourne la valeur de G associée au pixel noir et blanc.
- **retournerB :** Fonction virtuelle qui retourne la valeur de B associée au pixel noir et blanc.
- **retournerUneCopieProfonde :** Fonction virtuelle qui retourne une copie profonde du pixel.

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.

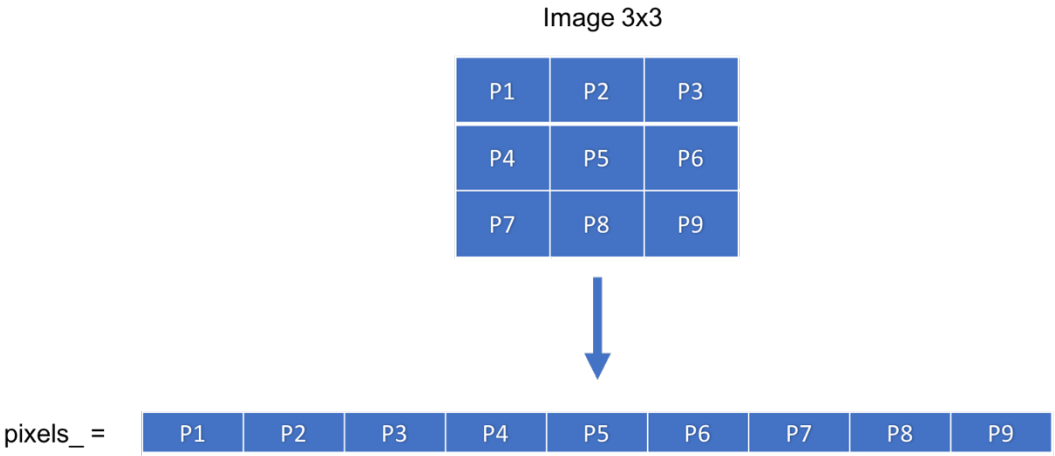


Figure 1 : Représentation d'une image après son stockage dans pixels_

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 modifiées :

- Un **constructeur par copie** (si nécessaire) qui doit copier les informations en accordance avec le nouveau tableau pixels_ et qui utilise les fonctions virtuelles.
- Un **constructeur par paramètre** qui utilise les nouvelles fonctions de lecture d'image qui profite des fonctions virtuelles (vous n'avez pas à implémenter les fonctions de lectures, elles sont fournies dans les fichiers du TP et sont déjà fonctionnelles).
- **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 les nouvelles fonctions virtuelles. N'oubliez pas de respecter la forme de pixels_ donnée à la figure 1.
- **convertirNB** qui convertit tous les pixels de l'image en PixelNB, sauf si l'image est déjà en noir et blanc (TypeImage ::NB). **Dois utiliser les fonctions virtuelles.**
- **convertirGris** qui convertit tous les pixels de l'image en PixelGris, sauf si l'image est déjà en gris (TypeImage ::Gris). **Dois utiliser les fonctions virtuelles.**
- **convertirCouleur** qui convertit tous les pixels de l'image en PixelCouleru, sauf si l'image est déjà en couleur (TypeImage ::Couleurs). **Dois utiliser les fonctions virtuelles.**

Les méthodes suivantes doivent être ajoutées :

- **toutMettreEnNegatif :** Fonction qui met tous les pixels d'une image en négatif. **Dois utiliser les fonctions virtuelles.**

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 :

- **Constructeur par copie :** (si nécessaire) Permets de créer une nouvelle instance d'un GroupeImage à partir d'un autre GroupeImage et d'en copier les données.
- **operateur= :** (si nécessaire) permet de copier les éléments d'un GroupeImage dans un autre GroupeImage.
- **ToutMettreEnNefatif :** Fonction qui permet de mettre toutes les images du groupe en négatif.

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 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. Peut-on instancier un objet de type Pixel?

Considérez la situation suivante :

```
Pixel* p1 = new PixelCouleur(20, 30, 50);
Pixel* p2 = new PixelBN(true);

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

2. Quelle sera la valeur de b et pourquoi?

3. Est-il possible de faire (*p1).obtenirR() et pourquoi?

Correction

La correction du TP se fera sur 20 points.

Voici les détails de la correction :

- (4 points) Compilation du programme ;
- (4 points) Exécution du programme ;
- (5 points) Comportement exact des méthodes du programme ;
- (2 points) Utilisation adéquate des fonctions virtuelle et du polymorphisme ;
- (1.5 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriée mémoire
- (1.5 points) Réponse aux questions.

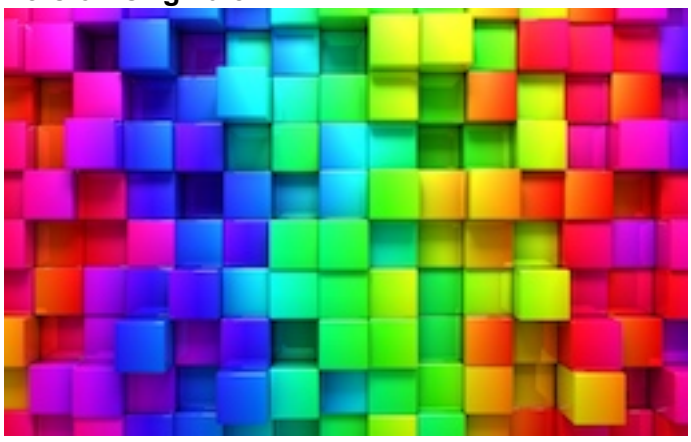
```
**************************************************
Creation du groupe d'image
**************************************************

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.
```

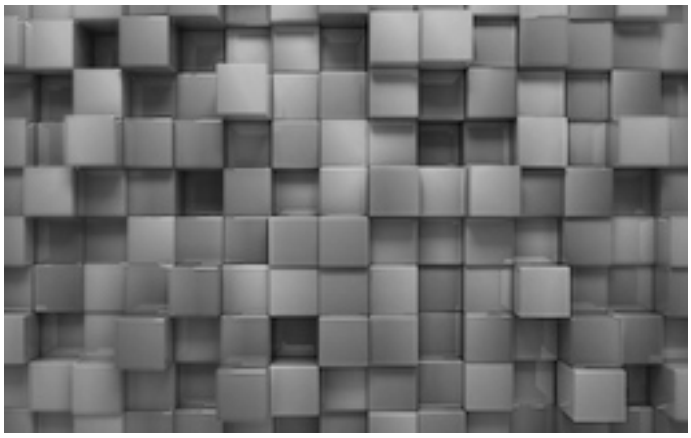
```
Sauvegarde de ./Ensemble d'images/Couleurs/im0Breaking-Bad.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im1Couleur.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im2Fall.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im3RM.bmp
Sauvegarde de ./Ensemble d'images/Couleurs/im4SolarEclipse.bmp
Sauvegarde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

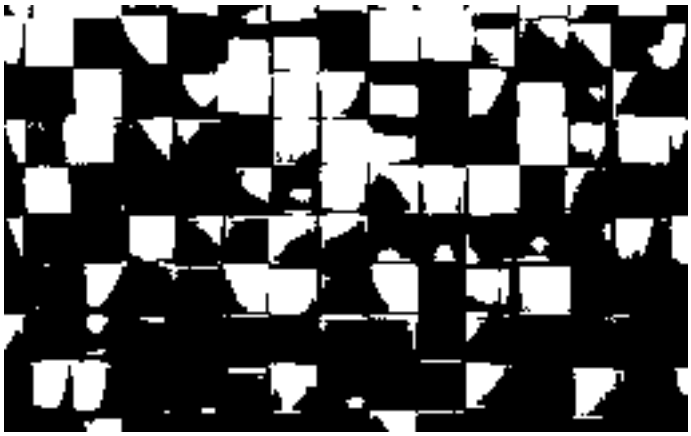
Version Originale



Version Nuances de Gris :



Version Noir et Blanc :



Version en négatif :