INF1010

Programmation Orientée-Objet

Travail pratique #2
Vecteurs et surcharge d'opérateurs

Objectifs :	Permettre à l'étudiant de se familiariser avec la surcharge d'opérateurs, les vecteurs de la librairie STL et l'utilisation du pointeur ***this***.
Remise du travail :	Lundi 2 octobre 2017, 8h
Références :	Notes de cours sur Moodle & Chapitre 14 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'entête, 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 entrainerait 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 gestion des images commencé au TP précédent en y intégrant les notions de vecteurs et de surcharge d'opérateurs.

Pour remplacer les tableaux dynamiques qui rendaient la gestion des pixels difficile, ce TP fait appel aux vecteurs de la librairie STL, soit std::vector. Et, pour faciliter les interactions avec les différents objets, la surcharge d'opérateurs sera utilisée.

Les vecteurs implémentés en C++ (STL) sont très pratiques : ce sont des tableaux dont la taille est dynamique. On peut y ajouter des éléments sans se préoccuper de la taille de notre vecteur étant donné que la gestion de la mémoire est automatique.

Le langage C++ est un langage avec lequel il est possible de redéfinir la manière dont fonctionnent la plupart des opérateurs (arithmétiques (+, -, *, /), d'affectation, etc..) pour de nouvelles classes. Nous pouvons donc redéfinir le comportement de ces opérateurs afin qu'ils effectuent une nouvelle opération ou englobent plusieurs opérations pour ces nouvelles classes.

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 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 TP1.

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 par la concentration des trois couleurs primaires soit le rouge le vert et le bleu du modèle RGB. On utilise les mêmes notations du TP précédent soit les caractères “R”, “G” et “B” pour désigner le rouge, vert et bleu. Un pixel ayant un mélange de plusieurs couleurs de base sera affiché avec le caractère “Q”(Voir les captures d’écran en référence)

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- La méthode qui retourne la couleur du pixel sous forme de caractère (R, G, B ou Q) .
- L’opérateur << (remplace la méthode d’affichage), qui affiche la couleur 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*.
- L’opérateur == qui prend un caractère en paramètre et permet de comparer un pixel avec sa couleur. Exemple (*pixel1*== ‘R’). L’opérateur return true si la couleur de pixel est notée par le pixel en paramètre.
- L’opérateur == de type *friend* qui permet d’inverser l’ordre de l’opérateur== précédent. Ainsi cet opérateur va pouvoir faire l’opération (caractere==pixel)

Classe Image

Cette classe caractérise une image par les attributs nomImage, nombrePixelEnHauteur, nombrePixelEnLargeur et pixels.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par copie (si nécessaire).
- L’opérateur = qui écrase les attributs de l’objet de gauche par les attributs de l’objet passé en paramètre.
- L’opérateur << (remplace la méthode de l’affichage), qui affiche une image suivant la forme du TP1
- 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*.
- L’opérateur == qui prend un nom en paramètre et compare une image avec une chaine 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*.

Classe *GroupImage*

Cette classe regroupe un ensemble d'images.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié

Cette classe contient l'attribut privés suivant :

- Images_ : Un vecteur de pointeurs Image, qui qui contiendra les différentes images

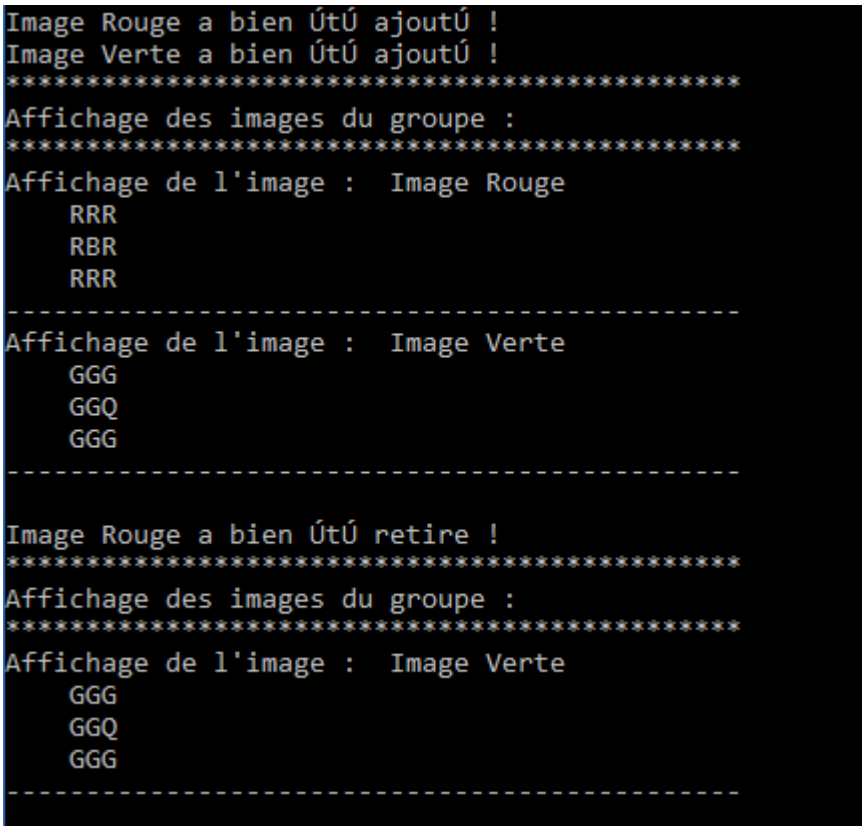
Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut
- Un destructeur
- Une méthode *ajouterImage()* qui permet d'ajouter une *image* reçu en paramètre. Deux images ne peuvent pas avoir le même nom. Pensez à utiliser l'opérateur == surchargé pour une *Image*.
- Une méthode retirerImage() qui permet de retirer l'image en utilisant le nom en paramètre. Pensez à utiliser l'opérateur == surchargé pour une *Image* .
- L'opérateur += qui prend en paramètre une image et qui l'ajoute au vecteur *images_* et qui retourne le groupe d'image après l'ajout de cet image.
- L'opérateur -= qui prend en paramètre une image qui l'enlève du vecteur d'images.
- L'opérateur << (remplace l'affichage), qui affiche les informations qui concerne un groupe d'images suivant l'exemple présenté à la fin du document.

Aide : Pensez à utiliser les différentes méthodes pour afficher les informations d'une image.

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 « user friendly » (ergonomique et joli) pour le rendu final
• Documenter votre code source
• Note : Lorsqu'il est fait référence aux valeurs par défaut, pour un string cela équivaut à la chaine vide, et pour un entier au nombre 0

Questions

1. Quelle est l'utilité de l'opérateur = et du constructeur par copie ?
2. Dans quel cas est-il absolument nécessaire de les implémenter ?
3. Qu'est-ce qui différencie l'opérateur = du constructeur par copie ?

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) Surcharge correcte des opérateurs ;
- (2 points) Utilisation correcte des vecteurs ;
- (1.5 points) Documentation du code ;
- (1 point) Utilisation correcte du mot-clé this pour les opérateurs ;
- (1 point) Utilisation correcte du mot-clé const ;
- (1.5 points) Réponse à la question.