INF1010

Programmation Orientée-Objet

Travail pratique #1
Allocation dynamique, composition et agrégation

Objectifs :	Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation
Remise du travail :	Lundi 25 Septembre 2017, 8h
Références :	Notes de cours sur Moodle & Chapitre 2-7 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.

Mise en contexte

La classification d'images est une pratique assez fréquente dans le domaine de l'imagerie. Elle consiste à regrouper des images suivant un ou plusieurs critères données. A titre d'exemple, dans le domaine d'imagerie médicale la classification automatique des scans des patients est très intéressant pour détecter les patients atteints d'une maladie donnée.

En tant que développeur stagiaire dans une compagnie de développement de logiciel d'imagerie, on vous demande de développer un système de classification d'images. Le but de ce TP sera ainsi de se familiariser avec les notions de bases de programmation orientée objet à travers l'exemple d'une application de classification d'images.

Vous serez amené ainsi à gérer des listes d'images et de faire des traitements simples sur des images qu'on détaillera tout au long du TP.

Travail à réaliser

Les fichiers *Image.h*, *Pixel.h* et *GroupImage.h* sont fournis. On vous demande de compléter les fichiers **.cpp* ainsi que de compléter le programme principal en suivant les instructions dans le fichier *main.cpp*.

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. Pour ce travail pratique, les caractères "R", "G" et "B" seront utilisés 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)

Cette classe contient les attributs privés suivants :

- tauxRouge (entier non signée)
- tauxVert (entier non signée)
- tauxBleu (entier non signée)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- Les méthodes d'accès aux attributs.
- Les méthodes *modifierTeinteRouge*, *modifierTeinteVert et modifierTeinteBleu* qui prennent en paramètre des entiers et qui permettent suivant le signe du paramètre

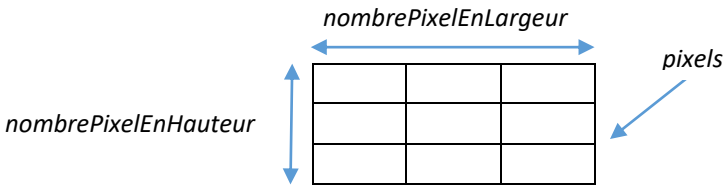
d'augmenter ou diminuer la concentration des couleurs primaires. Les concentrations minimales et maximales sont indiquées par des constantes dans le fichier *Const.h*

Classe Image

Cette classe caractérise une image.

Elle contient les attributs privés suivants :

- nomImage (string)
- nombrePixelEnHauteur (entier non signé)
- nombrePixelEnLargeur (entier non signé)
- pixels (Pixel**) qui contient l'ensemble des pixels de l'image.
  On opte ainsi pour la représentation suivante :



Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- Les méthodes d'accès aux attributs
- La méthode de modification de l'attribut nomImage.
- Les méthodes *doublerTailleEnLargeur*, *doublerTailleEnHauteur* qui permettent respectivement de doubler la taille de l'image en largeur et en hauteur.
- La *méthode augmenterTeintePixel(unsigned int positionLargeur, unsigned int positionHauteur, int increment, char couleur)* qui permet de modifier la couleur de la pixel de cordonnées (*positionHauteur, positionLargeur*). Il s'agit de modifier la concentration de la couleur donnée en paramètre en ajoutant la valeur de l'incrément.

Remarque : les pixels par défaut de l'image sont des pixels noirs (R=0, G=0 et V=0).

Classe *GroupImage*

Cette classe regroupe un ensemble d'images.

Elle contient les attributs privés suivants :

- type (string)
- nombreImages_ (entier non signé)
- capaciteImages_ (entier non signé)

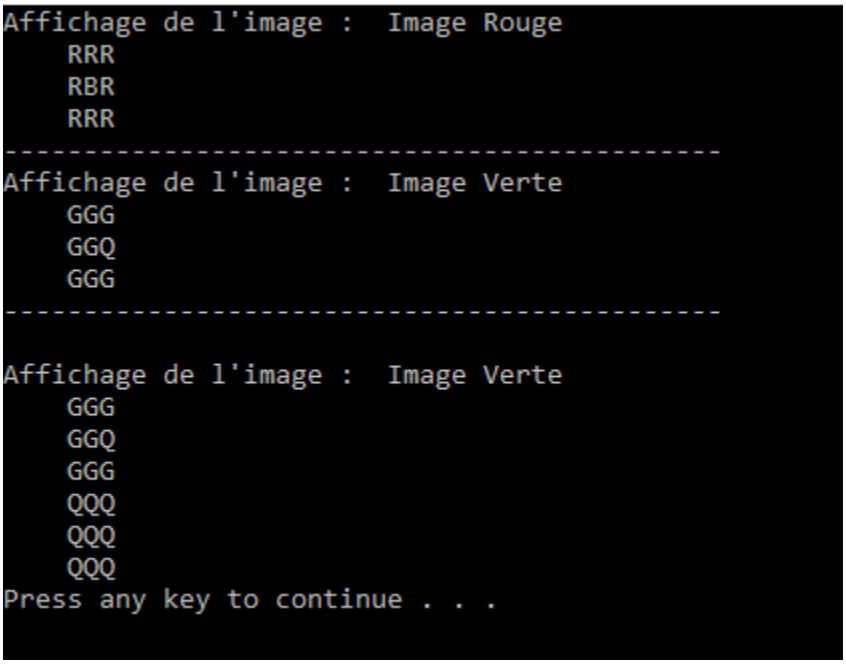
Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut
- Un destructeur
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- La méthode *modifierType(const string &type)* qui permet de modifier le type du groupe.
- La méthode *obtenirType()* qui retourne le type du groupe.
- La méthode *obtenirNombreImages()* qui retourne le nombre d'image du groupe.
- La méthode *obtenirImage(unsigned int indiceImage)* qui retourne l'image de l'indice *indiceImage*.
- Les méthodes *doublerTailleImageEnLargeur* et *doublerTailleImageEnHauteur* qui permettent respectivement de doubler la largeur et la hauteur de l'image d'indice donné en paramètre.

Main.cpp

Des instructions vous sont fournis dans le fichier Main.cpp et il vous est demandé de les suivre.

Votre affichage devrait avoir une apparence semblable à celle ci-dessous. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable



Spécifications générales

- Implémenter une méthode d'affichage dans chaque classe chaque fois que cela vous semble pertinent
- 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. Quel est le lien (agrégation ou composition) entre la classe *GroupImage* et la classe *Image* ? Justifiez.
2. Quel effet aura une méthode si elle aura la mention const.

Correction

La correction du TP1 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 ;

- (2 points) Documentation du code et bonne norme de codage ;

- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés ;

- (2 points) Utilisation adéquate des directives de précompilation ;

- (2 points) Allocation et désallocation appropriées de la mémoire ;

- (2 points) Réponse aux questions ;