TP$_1$ C++ – Image

Contraintes et remise des TPs

- **Chaque exercice doit être placé dans répertoire séparé (sauf consigne contraire), avec une Makefile permettant de le compiler.**
- **Chaque classe doit correspondre à un couple de fichiers .hpp/.cpp séparé** (sauf cas particuliers : classes template (pas de .cpp), classes très petites, etc.)
- Il n'est pas demandé d'utiliser un outil de documentation particulier (Doxygen, etc.) mais indentez et commentez votre code.
- De même, il n'est pas demandé de tests untaires, mais testez votre code dans un main. (L'utilisation d'un framework de test – Boost.Test, Catch, Google Test – donnera lieu à un bonus.)
- La correction tiendra compte de la briéveté des méthodes que vous écrivez (évitez les fonctions de plus de 25 lignes) ; n'hésitez pas à découper une méthodes en plusieurs sous-méthodes (privées) plus courtes.
- Pour l'UML si besoin, vous pouvez utiliser tout programme de votre choix, notamment l'outil `https://www.draw.io/`.
- Les noms de classe commencent par une majuscule.
- Les noms de méthodes et d'attributs commencent par une minuscule.
- La convention de nommage des accesseur est getNomAttribut() et setNomAttribut(...).
- Le code source et les éventuels autres documents (texte, diagrammes, etc.) doivent être envoyés sous forme d'une archive tar.gz [1].
- Le rendu est à déposer **sur le Moodle au plus tard 15 minutes après la fin du TP**.

Consignes spécifiques

Contrairement aux TPs Java, il n'est pour ce TP pas demandé de tests unitaire ou de style de documentation particulier. Vous devez néanmoins commenter votre code !

Pour le contrôle d'erreur, il vous est vivement conseillé d'utiliser les exceptions (vous pouvez utiliser `std::runtime_error` pour arrêter votre programme sur une erreur.)

Barème

Exercice 1 : 6 point. Exercice 2 : 10 points. Style, documentation et propreté du code : 4 points.

1 Exo 1 : Introduction (1h max)

1.1 Classe Vecteur

Programmez une classe `Vecteur` représentant un vecteur dans un espace bidimensionnel. On souhaite que cette classe possède deux attributs (privés) `x` et `y` représentant les coordon-

1. Pour faire une archive tar.gz : `tar zcvf mon_archive.tar.gz mon_repertoire`

nées, et les méthodes suivantes :

- Un constructeur par défaut créant un vecteur nul, et un constructuer permettant de donner des valeurs aux coordonnées ;
- Une méthode **void** affiche () qui affiche les coordonnées au format (x,y) ;
- Une méthode **float** norme() qui renvoie la norme du vecteur ;
- Une méthode Vecteur* addition(**const** Vecteur& v) qui renvoie un pointeur vers un nouvel objet vecteur correspondant à la somme du vecteur en cours et de v.
- Des accesseurs (getters const et non const) pour x et y ;

Vous devrez écrire les définitions dans un fichier vecteur.hpp, le code des méthodes dans un fichier vecteur.cpp, tester votre classe dans une fonction main, et fournir une makefile.

1.2 Copie, assignement, destructeur

Est-il nécessaire d'écrire un constructeur par copie, un opérateur d'assignement et un destructeur ? Justifiez votre réponse. Si oui, ajoutez le code de ces méthodes.

2 Exo 2 – Traitement d'images

On souhaite dans ce TP manipuler des images. Pour cela, nous allons utiliser une classe Image comportant :

- attributs :
  - la largeur (w, type size_t)
  - la hauteur (h, type size_t)
  - les données (data, type **unsigned char***)
- méthodes :
  - constructeur, qui prend en argument un nom de fichier et charge l'image
  - constructeur par copie
  - opérateur d'assignement
  - destructeur
  - load(**const** String& fichier), qui prend en argument un nom de fichier et charge l'image
  - write(**const** String& fichier), qui écrit l'image sur le disque dur dans le fichier dont le nom est donné en argument
  - get(**int** i, **int** j) : renvoie la valeur du pixel de coordonnées (i,j) (sans permettre de la modifier)
  - set(**int** i, **int** j, **unsigned char** valeur) : modifie la valeur du pixel de coordonnées (i,j)
  - get_w() : renvoie la largeur
  - get_h() : renvoie la hauteur

2.1 const

Pour écrire la classe ci-dessus, nous utiliserons les qualificateurs **const** de facon adaptée. Quelles méthodes doivent être const et non const ? Répondez en **justifiant votre réponse**.

2.2 Classe Image

Programmez la classe Image.

Puis, testez votre classe en faisant une fonction main qui charge une image pgm (exemples sur le site du cours) et l'écrit dans un autre fichier.

2.3 Classe Filtre

Un filtre est un objet qui modifie une image. La classe abstraite Filtre contient deux méthodes :

— **void** apply(Image& i) **const** : applique le filtre (méthode abstraite)
— **const** std:: string & get_name() **const** : renvoie le nom du filtre (stocké dans un attribut)

Programmez cette classe Filtre.

2.4 Classe FiltreAddition

Ecrivez une classe FiltreAddition qui dérive de la classe Filtre et ajoute une valeur constante positive ou négative (passée en argument du constructeur) à tous les pixels d'une image. (Dans le cas où un pixel devrait prendre une valeur négative ou supérieure à 255, il prend pour valeur 0 ou 255.) Elle sera donc équipée d'une méthode **void** apply(Image\& i) qui applique le filtre à l'image i.

Testez la classe dans votre fonction main.

2.5 Classe FiltreMoyenne

Le FiltreMoyenne remplace chaque pixel de l'image source par sa moyenne avec les pixels voisin sur une zone carrée donnée (zone de n par n, n étant un paramètre du filtre). Ecrivez la classe de facon similaire à FiltreAddition.

Testez la classe dans votre fonction main.

2.6 Classe SuiteDeFiltre

La classe SuiteDeFiltre dérive de la classe Filtre. Elle appelle successivement tous les filtres contenus dans un std : :vector<Filtre*>. Elle contient une méthode add(Filtre* f) qui permet d'ajouter des filtres dans la liste.

Testez dans votre fonction main.

3 Bonus

Implémentez les opérateurs suivant sur les images :

— + : addition de deux images
— + : addition d'un entier à une image
— - : soustraction de deux images
— - : soustraction d'un entier à une image
— () const : accesseur en lecture (remplace le get const)
— () : accesseur en écriture (remplace le set)

A Fichiers PGM

Une image PGM est une image en niveau de gris. Le format est le suivant :

P5
640 480
255
....données...

640 et 480 sont la largeur et la hauteur de l'image. Après le 255 commencent les données au format binaire (chaque octet code la valeur d'un pixel entre 0 et 255 comme un **unsigned char**).

B Lecture et écriture de fichiers

En C++, on utilise les classes ifstream et ofstream disponibles dans l'en-tête fstream.

B.1 Écriture

Documentation : http://www.cplusplus.com/reference/iostream/ofstream/

```
#include <fstream>

int main(int argc, char *argv[])
{
  std::ofstream ofs("/tmp/test.txt");
  if(!ofs.good())
    throw std::runtime_error("Unable to open file for writing");
  int x = 42;
  std::string h = "hello world";
  ofs << 42 << " " << h << std::endl; // equivalent printf
  unsigned char* buffer = new unsigned char[256];
  for (size_t i = 0; i < 256; ++i)
    buffer[i] = 42;
  ofs.write((char*)buffer, 256); // equivalent fwrite
  delete[] buffer;
}
```

B.2 Lecture

Documentation : http://www.cplusplus.com/reference/iostream/ifstream/

```
#include <fstream>

int main(int argc, char* argv[])
{
  std::ifstream ifs("testin.txt");
  if(!ifs.good())
    throw std::runtime_error("Unable to open file for reading");
  int x;
  std::string s;
  ifs >> x >> s; // lit x et s, equivalent scanf
  unsigned char * buffer = new unsigned char[256];
  ifs.read((char*)buffer, 256); // equivalent fread
  delete[] buffer;}
```