TP – Matrices (C++)

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**.

Barème

Classe Matrice : 15 points (fonctionnalités 12, documentation et gestion d'erreurs 3).
Classe Vecteur : 5 points (fonctionnalités 4, documentation et gestion d'erreurs 1)

1 Sujet

Le but de ce TP est de programmer une petite bibliothèque d'algèbre linéaire autour d'une classe Matrice équipée des opérateurs courants (+, -, *, etc.).

Le code devra être documenté par des commentaires et testé dans un main. Les erreurs (accès en dehors de la matrice, opérations entre matrices de mauvaises dimensions, trace d'une matrice non carrée, etc.) devront être détectées et lancer des exceptions (comme par exemple `std::runtime_error`).

1.1 Classe Matrice

Pour cette première partie, nous travaillerons avec des matrices de **double**. Voici ce que doit être capable de faire votre classe :

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

— constructeur qui construit une matrice nulle d'une taille donnée ;
— accès en lecture et écriture à la case (i, j) via l'opérateur () ;
— copie entre deux matrices (via l'opérateur = et le constructeur de copie) ;
— multiplication, addition et soustraction entre matrices ;
— multiplication, addition et soustraction d'un scalaire (ajouter un scalaire = l'ajouter à tous les coefficients de la matrice) ;
— trace d'une matrice carrée (somme des coefficients diagonaux) ;
— comparaison de matrices via l'opérateur == ;
— Méthode de classe renvoyant une matrice identité d'une taille donnée ;
— Méthode de classe renvoyant une matrice aléatoire (valeurs dans $[0; 1]$) d'une taille donnée ;

Vous devez par ailleurs écrire (en dehors de la classe) une fonction du type :
`std::ostream& operator<<(std::ostream&, const Matrice& m)`
permettant d'afficher la matrice dans un flux ostream.

Les tests unitaires ne sont pas demandés, mais vous devez fournir un main démontrant les capacités de votre classe.

Conseils :

— Testez votre code au fur et à mesure. Commencez par les méthodes de base nécessaires pour construire et manipuler la classe (constructeurs, destructeur, copie, accès à un élément) et testez-les dans le main. Puis, attaquez vous au reste de la classe (opérateurs mathématiques, ...)

1.2 Classe Vecteur

1.2.1 Bases

Dans un second temps, vous devez programmer une classe Vecteur (Matrice colonne) dérivée de Matrice qui ajoute les opérations suivantes :

— produit scalaire ;
— produit vectoriel (pour les vecteurs de taille 3) ;
— norme ;
— accès en lecture/écriture à l'élément i avec l'opérateur [].

1.2.2 Produit matrice-vecteur

Votre code devrait "naturellement" permettre de réaliser des produits matrice-vecteur, l'objet Vecteur étant considéré comme un objet Matrice (colonne) et pouvant être traité par l'**operator*** de Matrice (subsomption). Cependant, le résultat de cette opération est un également un objet Matrice (colonne), et non un objet Vecteur.

Ajoutez une autre fonction **operator*** à la classe Matrice, qui permet de réaliser des produits matrice-vecteur et renvoyant un objet Vecteur.

Note : Pour pouvoir utiliser la classe Vecteur dans les définitions des méthodes de `matrice.hpp`, il n'est pas possible de simplement inclure `vecteur.hpp` étant donné que celui-ci nécessite lui-même `matrice.hpp`. A la place, il faut faire une déclaration anticipée (*forward declaration*) de Vecteur dans `matrice.hpp` :

```
class Vecteur; // forward declaration
```

```
class Matrice {
        Matrice(size_t rows, size_t cols);
        ~Matrice();
        //...
        Vecteur operator*(/*... */) const;
};
```

Dans l'implémentation (matrice.cpp), il suffit ensuite d'inclure matrice.hpp.

1.3 BONUS : Généricité

Faites une copie de votre code avant de vous attaquer à ce bonus.

On veut maintenant avoir la possibilité de faire des matrices de divers types, par exemple d'entiers, de float, de double, ... sans avoir à définir plusieurs classes.

En utilisant les templates, transformez votre code en code générique permettant de gérer divers types de matrices et de vecteurs.

Note : Plutôt que de remplacer **double** par votre type template à la main partout, al est vivement conseillé d'utiliser la fonction "remplacer" de votre éditeur de texte ou IDE ! Puis de gérer à la main les éventuels soucis restants.