Projet : étape 2 (semaine 3 du semestre)

Buts

Nous allons cette semaine continuer la mise en place des premiers éléments de base de notre projet, en révisant (déjà !) la classe des vecteurs à 3 dimensions à la lumière de nos nouvelles connaissances (constructeurs et surcharge d'opérateurs).
Je vous demande également de commencer à utiliser la compilation séparée (eh oui, ça devait arriver !..).

Préliminaires :

Je vous rappelle qu'il est prévu une répartition des tâches du projet au sein du binôme. Je considère que chacun doit faire environ la moitié du travail lié au projet. Essayez donc de rapidement vous organiser, concevoir correctement la répartition et commencez à faire un Makefile.
Par ailleurs, **avant** de vous lancer directement dans la réalisation de la nouvelle partie du projet, je vous conseille de vous assurer de bien avoir compris les concepts présentés en cours, par exemple en faisant **au préalable** quelques exercices si nécessaire.
Cette première partie du projet (de semaine passé à la semaine prochaine) est assez progressive de sorte à vous permettre de vous organiser et prendre vos marques dans ce projet. Profitez-en donc pour faire les choses correctement dès le départ (cela permet de gagner du temps par la suite) et évitez absolument de prendre du retard... Je vous conseille d'avancer le plus possible en synchronisation avec le planning prévu.
Pour terminer avec les conseils et remarques préliminaires, n'oubliez pas de revenir de temps en temps consulter la page de description générale du projet et la page d'administration afin de situer votre travail courant par rapport au projet dans son ensemble.

[1+] Exercice P3 : la classe Vecteur3D revisitée

Le but de cet exercice est de modifier la classe Vecteur3D commencée la semaine passée en ajoutant les constructeurs et les principaux opérateurs.
Reprenez donc votre fichier Vecteur3D.cc (et éventuellement Vecteur3D.h et testVecteur3D.h).

P3.1 Constructeurs

Définissez les constructeurs suivants :

1. le constructeur par défaut qui crée un vecteur nul ;
2. un constructeur par coordonnées cartésiennes, prenant trois double comme arguments ;
3. le constructeur de copie, si vous pensez que cela est nécessaire.

[Question P3.1] Avez-vous ajouté un constructeur de copie ? Pourquoi (justifiez votre choix) ?
[Question P3.2] Pourquoi, contrairement à ce qui est trop souvent fait, l'écriture des deux premiers constructeurs avec une seule méthode en utilisant des valeurs par défaut aux arguments n'est-elle pas une très bonne idée ?
[Question P3.3] Si l'on souhaitait ajouter un constructeur par coordonnées sphériques (deux angles et une longueur)

- **a]** que cela impliquerait-il au niveau des attributs de la classe ?
- **b]** quelle serait la difficulté majeure (voire l'impossibilité) de sa réalisation en C++ ? (C'est d'ailleurs pour cela qu'on ne vous demande pas de faire un tel constructeur !)

Réfléchissez à ces questions (2b est difficile) et répondez-y dans votre fichier REPONSES.
Argumentez vos réponses.

P3.2 Opérateurs

Transformez les méthodes affiche et compare en des opérateurs.
[Question P3.4] Lesquels (opérateurs) ?
Répondez à cette question dans votre fichier REPONSES.
Modifiez votre programme de test *dans le même esprit* que ci-dessous (cela peut signifier que ce code n'est pas forcément 100% juste) :

```
Vecteur3D point1(1.0, 2.0, -0.1); // un vecteur en 3D
Vecteur3D point2;      // vecteur nul en 3D
Vecteur3D point3(point1); // copie

cout << "Point 1 :" << point1 << endl;
cout << "Point 2 :" << point2 << endl;

cout << "Le point 1 est ";
if (point1 == point2) {
    cout << "egal au";
} else {
    cout << "différent du" << endl;
}
cout << " point 2" << endl;

cout << "Le point 1 est ";
if (point1 != point3) {
    cout << "différent du" << endl;
} else {
    cout << "egal au";
}
cout << " point 3" << endl;
```

P3.3 Opérateurs algébriques

Continuons notre amélioration du code en utilisant la surcharge d'opérateurs pour les opérations algébriques de base définies la semaine passée :
transformez en des opérateurs toutes les méthodes de la classe Vecteur3D exceptée la norme euclidienne.
Pensez aux deux formes des opérateurs : la forme Op et la forme Op= ; par exemple, + et +=.
Modifiez votre programme de test de sorte à prendre en compte vos modifications (et testez votre programme).
Note : pour le produit vectoriel, vous pouvez utiliser l'opérateur operator^.

[2+] Exercice P4 : Modularisation

Il est peut être maintenant temps de se mettre à la compilation séparée... Voici un exercice censé vous aider à le faire.
L'idée est d'avoir un fichier séparé pour chaque classe plus un fichier pour le programme principal (et un/des fichier(s) pour les tests des classes).
Commençons par exemple avec la classe Vecteur3D.
Si vous ne l'avez pas encore séparée en trois fichiers, copiez le fichier Vecteur3D.cc en Vecteur3D.h. Et copiez-le encore une fois en testVecteur3D.cc. Vous avez donc maintenant trois fois le même fichier (même contenu) mais avec trois noms différents.
Nous allons évidemment maintenant modifier chacun de ces fichiers séparément.
Dans le fichier testVecteur3D.cc

1. supprimez toutes les définitions de la classe Vecteur3D et de ses méthodes ;
2. ne gardez que le main et les #include nécessaires ;
3. ajoutez #include "Vecteur3D.h".

Dans le fichier Vecteur3D.h

1. ne gardez que la définition de la classe Vecteur3D
   (garder aussi le #include <iostream> et le namespace pour le prototype de l'opérateur externe <<) ;
2. supprimez les définitions des méthodes (elles resteront dans le .cc)
   (Remarque (très) avancée : sauf celles que vous voulez garder inline, si jamais !) ;
3. ajoutez
   ```
   #ifndef PRJ_VECTEUR_H
   #define PRJ_VECTEUR_H
   ```
   en **tout début** de fichier, et
   ```
   #endif // PRJ_VECTEUR_H
   ```
   tout à la fin.
   Ceci sert à éviter qu'un même fichier .h soit inclus par mégarde plusieurs fois dans une même compilation (ça arrive ... surtout quand votre projet va augmenter de taille !).
   Le nom PRJ_VECTEUR_H est bien entendu totalement arbitraire et peut être changé, **mais** il doit être trois fois le même ci-dessus, et **surtout** bien choisi pour être **unique** à

chaque fichier .h différent.

Dans le fichier Vecteur3D.cc

1. Ne gardez que les définition des méthodes ;
2. « Extériorisez » (hors de la classe elle-même) leur définitions (en ajoutant Vecteur3D:: devant le nom de la méthode (sauf pour les opérateur externes bien sûr !) ;
3. ajoutez #include "Vecteur3D.h".

Créez le fichier Makefile permettant de construire (make) votre projet à partir de ses constituants.
Je vous donne ci-dessous un exemple simple, mais vous recommande néanmoins d'aller voir le tutoriel sur les Makefile pour en savoir plus :

```
CC  = c++
CXX = c++

# Partie commentée : choisissez les options que vous voulez avoir
#                    en décommentant la/les lignes correspondantes
#
# CXXFLAGS  = -ansi -pedantic -Wall   # pour les purs et durs
# CXXFLAGS += -g                      # pour debugger
# CXXFLAGS += -pg                     # pour profiler
# LDFLAGS  += -pg                     # pour profiler
# CXXFLAGS += -O2                     # pour optimiser la vitesse

all:: testVecteur3D

Vecteur3D.o: Vecteur3D.cc Vecteur3D.h

testVecteur3D.o: testVecteur3D.cc Vecteur3D.h

testVecteur3D: testVecteur3D.o Vecteur3D.o
```

Une fois tout ceci fait, tapez simplement make (dans un terminal se trouvant dans bon répertoire (~/cpp/projet)) et...
..oh miracle ! Ça marche. (enfin j'espère pour vous !)
Vérifiez en lançant votre programme : ./testVecteur3D
