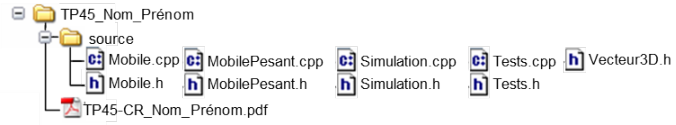
TP de C++ n° 4

Simulation de mobiles (première partie)

Ce TP se poursuivra sur la prochaine séance de TP, et vous n'avez rien à rendre à l'issue de cette séance. Il vous est demandé de façon générale de bien structurer le code, avec pour chaque classe un fichier `.h` *et un fichier* `.cpp`*, et votre fonction* `main` *à part faisant appel aux différents tests. Le résultat devra se présenter sous la forme d'une archive qui, une fois extraite, doit révéler l'arborescence suivante :*



où aucun fichier ne doit comporter de fonction `main`*, où le fichier* `tests.cpp` *doit contenir tous les tests demandés, et où le fichier* `.pdf` *doit être un compte-rendu au format PDF présentant le travail réalisé et incluant des indications de conception, d'implémentation, et une justification des tests vous permettant d'affirmer que le code ne présente aucun défaut.*

— : —

Le but de ce TP est de réaliser l'infrastructure d'un logiciel de simulation de mobiles en trois dimensions. De façon générale, il vous est demandé :

- — d'utiliser le passage des arguments par référence, selon ce qui a été indiqué en cours ;
- — de faire attention aux fuites de mémoire (un `delete` pour un `new`) — pensez à utiliser valgrind ;
- — utilsier le mot clef `const` chaque fois que c'est utile, pour les arguments des méthodes comme pour les méthodes elles-mêmes ;
- — de ne pas mettre d'attributs publics (sauf dans la classe `Vecteur3D`), à moins d'avoir une justification pour cela.

— : —

Exercice 1 : ***Mobiles***

On se munit d'une classe `Vecteur3D` selon ce modèle (à recopier dans `Vecteur3D.h`) :

```
class Vecteur3D {
   public:
      double x, y, z ;
} ;
```

On pourra enrichir un peu cette classe si besoin en ajoutant des constructeurs dans le `.h` ci-dessus.

1°) Créer une classe `Mobile`. Un mobile a un `nom` (`std::string`), une `position` (`Vecteur3D`) et une `vitesse` (`Vecteur3D`). Créer constructeurs, destructeur, accesseurs, méthodes d'affichage.

2°) Un mobile évolue selon une règle simple. Si $\vec{X}$ est sa position et $\vec{V}$ sa vitesse, alors sa nouvelle position au bout de $\mathrm{d}t$ secondes est : $\vec{X}+\mathrm{d}t\times\vec{V}$, et sa vitesse reste inchangée. Doter la classe mobile d'une méthode `avance(double dt)` qui implante cette règle de movement.

3°) Tester la classe dans une fonction `bool testMobile1()` qui renvoie `true` si les tests passent. Justifier les tests dans le compte-rendu.

Exercice 2 : ***Simulation***

1°) Créer une classe `Simulation` qui contienne un `temps` courant (`double`) et gère une liste de pointeurs sur des mobiles `std::list<Mobile*> corps`.

2°) Doter cette classe de trois méthodes :
- `ajoutCorps(Mobile *)` qui ajoute un mobile,
- `oteCorps(Mobile *)` qui supprime un mobile,
- `afficheCorps()` qui affiche la liste des mobiles.

3°) Munir cette classe d'une méthode `simuler(double dt)` qui parcourt la liste des mobiles en appelant leur méthode avance puis modifie le temps courant en lui ajoutant `dt`.

4°) Tester la classe dans une fonction `bool testSimulation1()` qui renvoie `true` si les tests passent. Justifier les tests dans le compte-rendu.

Exercice 3 : ***Gestion de la mémoire***

1°) Ajouter un destructeur dans la classe `Simulation`. Celui-ci doit s'assurer que tous les mobiles présents dans `corps` sont bien détruits.

2°) Qu'est-ce que cela implique pour l'utilisateur de la classe `Mobile` ? N'y a-t-il pas un problème si des mobiles sont créés à la fois par allocation dynamique et comme par allocation automatique lors de la déclaration de la variable ?

3°) Tester le destructeur dans une fonction `bool testSimulation2()` qui renvoie `true` si les tests passent. Expliquer le problème et justifier les tests dans le compte-rendu.

Exercice 4 : ***La gravité***

1°) On veut pouvoir simuler des corps soumis à la gravité. Créer une classe `MobilePesant` dont les objets ont une `masse` (`double`). Établir son lien (évident) avec `Mobile`.

2°) Un corps soumis à la gravité évolue selon la loi suivante : si $\vec{X}$ est sa position et $\vec{V}$ sa vitesse, alors, au bout de $\mathrm{d}t$ secondes, sa nouvelle position est : $\vec{X} + \mathrm{d}t \times \vec{V}$, et sa nouvelle vitesse est : $\vec{V} + \mathrm{d}t \times \vec{g}$ où g est un vecteur constant de coordonnées (0 0 -9,81). Faire le nécessaire pour prendre en compte cette loi de mouvement dans la méthode `avance`.

3°) Tester la classe dans une fonction `bool testMobile2()` qui renvoie `true` si les tests passent. En particulier, réaliser le test suivant : un mobile pesant lâché d'une hauteur h doit mettre $\sqrt{\frac{2h}{9,81}}$ pour attendre le sol ; retrouver ce résultat en appelant la méthode avance plusieurs fois par petits pas de temps. Justifier les tests dans le compte-rendu.

Exercice 5 : ***Simulation avec mobiles pesants***

1°) La classe `Simulation` doit être capable sans travail supplémentaire de gérer mobiles ordinaires ou pesants. Le vérifier en réalisant un test récapitulatif général : au moins un mobile ordinaire et un mobile pesant dans la liste, une boucle sur le temps qui appelle plusieurs fois la méthode `simuler`, et des affichages et des vérifications des valeurs attendues. Mettre ces tests dans une fonction `bool testSimulation3()` qui renvoie `true` si les tests passent. Justifier les tests dans le compte-rendu.

2°) Compléter la classe `Simulation` par un constructeur par copie : celui-ci doit notamment copier tous les mobiles de la liste `corps`. Il y a cependant un problème : comment savoir si le mobile à copier est un `Mobile` ou un `MobilePesant` ? Pour résoudre ce problème, la solution la plus classique consiste à créer dans la classe `Mobile` une méthode virtuelle `Mobile* copie()`, sans argument, qui retourne un pointeur obtenue par appel du constructeur par copie : il faudra implanter cette méthode dans chaque classe et sous-classe [1]. Tester ce constructeur par copie dans une fonction `bool testSimulation4()` qui renvoie `true` si les tests passent. Justifier les tests dans le compte-rendu.

1. Si vous préférez, c'est un peu un moyen détourné de rendre virtuel le constructeur par copie des mobiles...