1 Objectifs du TD

1. Comprendre la surcharge d'opérateurs.
2. Différencier les surcharges internes et externes.
3. Découvrir les exceptions.

Le but de ce TP est de créer une classe vecteur avec des opérateurs adaptés.

1. Déclarez une classe `Vector` avec deux champs publics (pour l'instant) : un tableau de réels `vect` et un entier positif `size`. Cette classe doit aussi contenir un constructeur à paramètre un entier positif fixant la taille du tableau et initialisant tous les éléments du tableau à 0, un constructeur de copie et un destructeur.
2. Définissez les méthodes de cette classe.

Afin de pouvoir accéder aux éléments du vecteur et les modifier, nous allons surcharger l'opérateur `[]`. Ajoutez à `Vector.h` les lignes suivantes (en public) :

1. `double operator[](unsigned int) const;`
2. `double& operator[](unsigned int);`

La premiére doit permettre simplement d'accéder aux éléments du tableau, la seconde de les modifier. Définissez ces opérateurs dans `Vector.cpp` : ils renvoient simplement l'élément à la i-ième ligne du vecteur.

Surchargez les opérateurs `+=` et `-=` afin qu'ils permettent respectivement d'ajouter et de soustraire un réel à chacun des éléments du vecteur. Ces opérateurs doivent renvoyer le vecteur une fois modifié. On utilisera pour cela le pointeur `this`.

Une autre facon de surcharger les opérateurs est de le faire hors de la déclaration de la classe.

1. Ajouter dans `Vector.h`, mais hors de la définition de la classe, la ligne suivante : `Vector operator+(const Vector &,const Vector &);`. Définissez ensuite l'opérateur afin qu'il effectue la somme de des deux vecteurs en paramétres.
2. Surchargez de même l'opérateur `-`.
3. Surchargez de même l'opérateur `<<`. La déclaration sera la suivante : `std::ostream& operator<<(std::ostream &,const Vector &);`.

   `ostream` est le type des flux sortant. L'opérateur surchargé doit ajouter au flux en paramétre les flux nécessaires pour que l'affichage du flux obtenu écrive le vecteur à l'écran sous la forme `[1,2,4,3]`.

Changez la déclaration de la classe afin que les champs soient maintenant privés. Que se passe-t-il lorsque vous compilez ? Quels opérateurs posent-ils problème ? Afin de régler ce problème il faut déclarer ces opérateurs comme méthodes amies de la classe `Vector`. Ajoutez pour cela leur déclaration précédée du mot clé `friend` au début de la classe.

Ecrivez le fichier `main.cpp` et testez les opérateurs surchargés.

Il faut éviter que l'utilisateur tente d'accéder à un élément hors du tableau ou fasse la somme ou la différence de deux tableaux de taille différentes. Le programme génerera pour cela une exception à l'aide de `throw()`. Toute ligne de commande suceptible de produire une erreur sera placée dans un bloc `try{}` et un bloc `catch{}` placé juste après le preécédent permettra de gérer cette erreur (ici: prévenir l'utilisateur).

1. Incluez toutes les lignes de commande de votre `main`, excepté le `return` final, entre les accolades d'un `try{}`.

2. Ajoutez à la suite de ce bloc `try{}` un bloc `catch(string & message){}` qui affiche (entre ses accolades) un message d'erreur. Le paramétre `message` est une chaine de caractère décrivant le type d'erreur produite.

3. Modifiez `Vector.cpp` afin que chaque fois qu'une erreur se produit la ligne de commande `throw("Message d'erreur")` soit exécutée, sauvant ainsi l'utilisateur d'une erreur fatale. L'affichage se fera à l'aide de `cerr` à la place de `cout` (la syntaxe est identique).