Sommaire

La Classe Vecteur

Introduction

Dans ce TP, nous nous proposons de développer une classe C++ nommée `Vecteur` permettant de traiter des vecteurs de $\mathbb{R}^n$. Dans quelques temps, nous généraliserons cette classe en un patron de classe `Vecteur<>` pour faire par exemple des vecteurs de `Ratio`.

Travail

N'hésitez pas durant votre travail à vous rafraîchir au besoin la mémoire à propos de vecteur et de géométrie vectorielle grâce à Wikipedia (voir liens).

Donnez la définition de la classe `Vecteur` répondant au cahier des charges suivant. Toutes les fonctionnalités demandées doivent être réalisées, et leur bon fonctionnement vérifié.

La classe `Vecteur` code un vecteur de n éléments qui sont tous des réels double précision. On pourra créer des `Vecteur` de dimension quelconque.

Il doit être possible de définir une instance de `Vecteur` en précisant sa dimension et une valeur qui par défaut sera 0 pour initialiser chaque coordonnée.

Il doit également être possible de créer une instance de `Vecteur` sans rien préciser, ce qui fera - par convention - un vecteur "vide", c'est-à-dire un vecteur en dimension 0, sans coordonnées. Attention, à bien mettre à `NULL` le pointeur sur le tableau dans ce cas là pour indiquer qu'aucun tableau n'a été alloué.

Remarque : il est légal de faire "`delete [] p;`" où `p` est `NULL` : cela ne fait rien du tout.

Il doit également être possible (avec un autre constructeur) de créer un vecteur à partir de sa dimension n et d'un tableau constant de n réels reçu en paramètre.

Comme pour toute classe que vous écrirez, vous devez définir le constructeur de copie pour un `Vecteur`.

Vous devez, comme vu en cours, réaliser l'operator= qui permet la copie d'un vecteur dans un autre. Attention, l'objet destination de la copie n'est pas vide, il faut penser à détruire le tableau de coordonnée d'origine, puis en recréer un nouveau de la bonne taille, et enfin recopier les coordonnées.

L'accès à la $i^{ième}$ coordonnée du vecteur *v* se fera grâce à deux fonctions accesseurs que vous devrez réaliser en utilisant deux surcharges de l'opérateur `operator[]`. Évidement, la valeur reçue en paramètre de l'opérateur indiquera le rang de la coordonnée à manipuler, en commençant comme toujours en C++ à compter à partir du rang 0 qui correspond la première coordonnée. La première surcharge aura pour signature `double Vecteur::operator[](unsigned int i ) const` ; elle permet de consulter la valeur d'une coordonnée tout en garantissant (grâce au `const`) que le Vecteur lui-même n'est pas modifié. La deuxième surcharge aura pour signature `double& Vecteur::operator[](unsigned int i)` ; elle permet la modification d'une coordonnée du Vecteur. (En fait elle permet aussi la consultation de la coordonnée, mais elle ne permet pas de garantir la constance du Vecteur sur lequel on l'applique, c'est pour cela qu'il faut écrire séparément la première fonction).

Une méthode `double norme()` renvoie la valeur de la norme euclidienne du vecteur. On rappelle que la norme euclidienne d'un vecteur est égale à la racine carrée de la somme du carré de chaque coordonnée.

Une méthode `Vecteur normalise()` permet de fabriquer un nouveau vecteur obtenu en divisant les coordonnées de celui sur lequel on applique la fonction par sa norme. On obtient donc un nouveau `Vecteur` dont la norme est égale à 1.

L'écriture d'un vecteur sur un flux (avec <<) doit produire une sortie de la forme "`<3:3.2,0,1>`", désignant un vecteur de 3 réels dont la première coordonnée est 3.2, la deuxième 0 et la troisième 1.

La lecture sur un flux (avec >>) doit permettre de lire un vecteur de la même forme. Attention, la lecture peut (doit ?) modifier la dimension du vecteur, et donc il sera a priori nécessaire de réallouer un autre tableau de coordonnée (un peu comme dans le constructeur de copie ou dans le `operator=`). La lecture d'autre chose qu'un vecteur bien formé (c'est-à-dire n'utilisant pas la forme "`<3:3.2,0,1>`") doit laisser le flot dans un état d'échec.

Quelques raisons d'échec peuvent être :

- l'absence des symboles de formatage prévus "<", ":", "," et ">" ;
- trop ou trop peu de coordonnées par rapport à la dimension indiquée ;
- etc.

On programmera la multiplication d'un vecteur par un réel quelconque (à droite ou à gauche !).

On programmera un `operator-` (unaire) permetant de calcul de l'opposé du vecteur sur lequel on applique l'opérateur, c'est-à-dire le vecteur obtenu en prenant l'opposé de chaque coordonnée.

On programmera la somme et la soustraction de deux vecteurs de même dimension. Toute tentative de réaliser une somme (ou soustraction) de deux vecteurs de dimensions différentes doit conduire au déclenchement d'une exception.

On programmera une méthode `Vecteur appliquer1(double (*f)(double)) const` qui permet de calculer un vecteur (résultat) dont les coordonnées sont obtenues en appliquant la fonction *f* à chaque coordonnée du vecteur sur lequel la méthode est appelée.

On implémentera le produit scalaire de deux vecteurs grâce à l'`operator*`. On rappelle que le produit scalaire de deux vecteur *v* et *w* est le réel qui est obtenu en faisant la somme des $vi \times wi$ pour tout les *i*.

Question

Pouvez-vous implémenter le produit vectoriel avec l'`operator*` ? Pourquoi ? (Contrôlez auprès de l'enseignant surveillant le TP que vous avez trouvé la bonne raison). L'implémentation du produit vectoriel est demandé uniquement en question subsidiaire, c'est-à-dire en bonus.

Remarques

- Avez-vous mis des `const` à tous les endroits où leur présence serait justifiée ?
- Avez-vous vérifié qu'un `delete` est déclenché en réponse à chacun des `new` de votre programme ?
- Êtes-vous certain d'avoir réalisé les constructeurs et opérateurs nécessaires ?

Si oui, tentez de répondre à la question subsidiaire...

Question subsidiaire

Ajoutez une méthode pour le calcul du produit vectoriel.

Généralisez la fonction membre `appliquer1()` en une fonction non-membre `Vecteur appliquer2(double (*f2)(double, double),const Vecteur& v1, const Vecteur& v2 )` permettant d'appliquer une fonction de deux variables sur les coordonnées de deux vecteurs.

Liens

- Page du cours de programmation C/C++ d'Éric REMY :

http://www.iut-arles.up.univ-mrs.fr/eremy/Ens/Info1.C++/index.html

- Planche de TP n°2 d'Éric REMY :

http://pluton.up.univ-mrs.fr/eremy/Ens/LPSIL.C++/tp2/index.html

- Pour approfondir vos connaissances en C et C++ :

http://www.iut-arles.up.univ-mrs.fr/eremy/Ens/Info1.C++/approfondir.html

- Vecteur : http://fr.wikipedia.org/wiki/Vecteur

- Calcul vectoriel en géométrie euclidienne : http://fr.wikipedia.org/wiki/G%C3%A9om%C3%A9trie_vectorielle

Validation

```
long double f1(long double);

long double f2(long double, long double);

int main() {
   long double t[5] = {1.0, 2.0, 3.0, 4.0, 5.0};

   Vecteur v0, v1(4, 0.0), v2(5, t);
   cout.precision(2);
   //cout << "V0 = " << setprecision(1) << v0 << endl;
   cout << "V0 = " << v0 << endl;
   cout << "V1 = " << v1 << endl;
   cout << "V2 = " << v2 << endl;
   v0 = v2;
   cout << "V0 = " << v0 << endl;
   cout << "Norme de V0 = " << v0.norme() << endl;
   v0.normalise();
   cout << "V0 = " << v0 << endl;
   cout << "Norme de V0 = " << v0.norme() << endl;
   cout << "Entrez un vecteur : ";
   cin >> v0;
   if (! cin) {
      cout << "ERREUR de lecture\n";
      exit (-1);
   }
   cout << "V0 = " << v0 << endl;
   cout << "V0 * 2 = " << v0 * 2 << endl;
   cout << "2 * V0 = " << 2 * v0 << endl;
   cout << "-V0 = " << -v0 << endl;
   cout << "V0 + V1 = ";
   try {
       cout << (v0 + v1) << endl;
   }
   catch (Erreur & e) {
      cout << e.what() << endl;
   }
   v0 = Vecteur(3, t);

   cout << "V0 = " << v0 << endl;
   cout << "V0.appliquer1(f1) = " << (v1 = v0.appliquer1(f1)) << endl;
   cout << "V0 = " << v0 << "V1 = " << v1 << endl;
   try {
      cout << "appliquer2(f2, V0, V1) = " << appliquer2(f2, v0, v1) <<
endl;
   }
   catch(Erreur & e) {
      cout << e.what() << endl;
   }
```

```
   cout << "V0 ^ V2 = " << v0 << " ^ " << v2 << " = ";
   cout << "V0 ^ V2 = " << v0 << " ^ " << v2 << " = ";
   v2 = Vecteur(3, t);
   cout << "V0 ^ V2 = " << v0 << " ^ " << v2 << " = ";
   try {
      cout << (v0 ^ v2) << endl;
   }
   catch (Erreur & e) {
      cout << e.what() << endl;
   }
   return 0;
}

long double f1(long double x) {
   return x / 2;
}

long double f2(long double x, long double y) {
   return (x + y) / 2;
}
```