TP n°10 : Un vecteur N-d

Préparation.

Lancez eclipse et créez un nouveau projet C++ : C++ Managed Build (type : *Empty*, Toolchain : *Linux*, Config : *Release*). Nommez ce projet Vecteur. *En dehors d'eclipse*, désarchivez le fichier vecteur.tar dans le répertoire du projet.

Retournez sous eclipse et cliquez avec le bouton droit sur votre projet Vecteur, sélectionnez l'action *refresh/actualiser* qui devrait faire apparaître quatre fichiers dans votre projet.

But du TP

Il s'agit de concevoir, en C++, une classe de gestion de vecteurs (au sens mathématique du terme) en N dimensions. Pour concevoir ce TP, nous utiliserons quatre fichiers :

- « memory.cxx » : fichier contenant une redéfinition des allocateurs mémoire. Ce fichier ne sera pas à modifier.
- « vecteur.h » : fichier contenant la déclaration de la classe *Vecteur* ainsi que des fonctions qui l'utilisent. Attention, ce fichier ne doit contenir aucune implémentation.
- « vecteur.cpp » : fichier contenant l'implémentation des méthodes et fonctions déclarées dans le fichier « vecteur.h ».
- « main.cpp » : fichier contenant le programme principal.

Dans la suite de ce TP, il vous est demandé de mettre des commentaires pour toute méthode / fonction déclarée. Ces commentaires devront décrire les traitements effectués, la signification des paramètres et l'ensemble des préconditions.

Afin d'éviter toute erreur de programmation, il vous est aussi demandé d'utiliser la fonction *assert* (définie dans le fichier « cassert »), afin de vérifier toutes les préconditions des méthodes et fonctions.

Attention, dans le sujet de ce TP, plusieurs prototypes vous sont fournis ; vous devez **impérativement** les respecter.

Le fichier « memory.cxx » redéfinit le système d'allocation mémoire. Grâce à ce système, un certain nombre de messages sont affichés à chaque demande d'allocation / désallocation mémoire. D'autre part, lorsque le programme se termine et qu'il reste de la mémoire non désallouée, cela vous est signalé.

Partie 1 : Conception de la classe Vecteur

Attention : dans cette partie du TP, il vous est demandé de ne **jamais** faire d'affectation de vecteurs. Autrement dit, soit *a* et *b* deux vecteurs, une instruction du type *a=b* ne doit **jamais** être utilisée.

Question 1 : Concevez une classe nommée *Vecteur* représentant un vecteur à N dimensions à coordonnées réelles (*float*). Cette classe devra être dotée des fonctionnalités suivantes (réfléchissez aux attributs) :

- Constructeur avec deux paramètres permettant de fournir le nombre de dimensions du vecteur et une valeur initiale à affecter à toutes les coordonnées.
- Constructeur avec un paramètre permettant de fournir le nombre de dimensions du vecteur et initialisant les coordonnées à 0.0.
- Constructeur sans paramètre construisant un vecteur de dimension 3 et initialisant les coordonnées à 0.0.
- Une méthode (nommée get) permettant de consulter la valeur d'une coordonnée.
- Une méthode (nommée set) permettant de changer la valeur d'une coordonnée.
- Une méthode (nommée dimensions) permettant de connaître le nombre de dimensions du vecteur.

Attention : la déclaration de la classe doit se faire dans le fichier « vecteur.h » et l'implémentation de toutes les méthodes dans le fichier « vecteur.cpp ». Pour le moment, **ne programmez pas** le destructeur de cette classe. Dans le constructeur, affichez un message signalant la construction du vecteur ainsi que ses caractéristiques.

Question 2 : Nous allons programmer des *fonctions* pour effectuer la saisie et l'affichage d'un vecteur. Ces fonctions **devront** avoir le prototype suivant :

- void afficherVecteur(const Vecteur * v, std::ostream & out = std::cout) : fonction qui affiche le vecteur v dans le flux de sortie out.
- Vecteur * lireVecteur(std::istream & in = std::cin) : fonction qui crée et initialise un vecteur dont les caractéristiques (nombres de dimensions et valeur de chaque coordonnée) sont fournies par l'utilisateur sur le flux d'entrée in.

***Attention* :** les prototypes de ces fonctions doivent être placés dans le fichier vecteur.h **après** la fermeture de la classe Vecteur ; les définitions seront placées dans le fichier vecteur.cpp.

Question 3 : Faites le programme principal dans le fichier « main.cpp ». Ce programme devra demander à l'utilisateur de saisir un vecteur et ensuite l'afficher. Dans un second temps, ce programme devra demander à l'utilisateur de modifier l'une des coordonnées du vecteur. Lors du test, essayez de fournir une coordonnée dans une dimension supérieure à celle du vecteur. Que se passe-t-il ? (bien évidemment, cette observation n'est possible que si vous avez respecté toutes les consignes fournies dans ce TP, notamment, l'utilisation de la fonction *assert*).

Attention: Avant de traiter les questions suivantes, positionnez l'option de compilation suivante : -fno-elide-constructors (voir les propriétés du projet, C/C++ Build, Settings, C++ Compiler, Optimization) ; en voici la signification (*man g++*).

-fno-elide-constructors : The C++ standard allows an implementation to omit creating a temporary that is only used to initialize another object of the same type. Specifying this option disables that optimization, and forces G++ to call the copy constructor in all cases.

Question 4 : Programmez une fonction permettant d'additionner deux vecteurs. Cette fonction **devra** avoir le prototype suivant :

- Vecteur add(const Vecteur * v1, const Vecteur * v2)

Une fois cette fonction réalisée, modifiez votre programme principal afin d'afficher la valeur de la somme de deux vecteurs saisis par l'utilisateur.

Question 5 : Comme vous avez pu le remarquer, lorsque vous quittez votre programme, le gestionnaire de mémoire vous affiche des erreurs signalant un certain nombre de zones mémoire non désallouées. Programmez le destructeur de la classe *Vecteur* afin de résoudre ce problème. Testez votre programme.

Question 6 : Comme vous pouvez le remarquer, la programmation de ce destructeur ne résout pas entièrement votre problème. D'une part, l'allocateur mémoire affiche toujours des erreurs (d'ailleurs, parmi celles-ci, une nouvelle erreur est apparue), d'autre part, le résultat de l'addition de vos deux vecteurs ne doit plus être correct (cela vient de l'erreur précédente). D'où vient ce problème ? Comment le résoudre ? Programmez la fonctionnalité manquante.

Avant de passer à la suite, assurez-vous que votre programme ne provoque aucun fuite mémoire.

Partie 2 : Redéfinition d'opérateurs

Question 7 : Modifiez votre programme principal afin d'utiliser une affectation entre vecteurs. De nouveau, l'allocateur mémoire doit faire apparaître un certain nombre de dysfonctionnements. Corrigez ce problème.

Question 8 : Pour additionner deux vecteurs, nous utilisons, actuellement, la fonction *add.* Surchargez l'opérateur d'addition (+) sous la forme d'une méthode de la classe *Vecteur* et modifiez votre programme principal afin de tester cette opération.

Remarques :

- réfléchissez au type des paramètres de cet opérateur ;
- dans le programme client, il faudra pouvoir additionner au moins trois vecteurs dans une même expression sans provoquer de fuite de mémoire ; on doit donc pouvoir écrire :

  Vecteur somme (v1 + v2 + v3) ; où v1, v2 et v3 sont de type Vecteur
- expliquez (en vous servant de l'exemple précédent) pourquoi cet opérateur ne doit pas renvoyer son résultat par référence ni pointeur.

Question 9 : Nous souhaitons calculer le *produit scalaire* de deux vecteurs. Surchargez l'opérateur de multiplication (*) afin qu'il permette de calculer ce produit scalaire, mais contrairement à la question précédente, cette surcharge doit être effectuée par une fonction et non une méthode de la classe *Vecteur*. Modifiez votre programme principal afin de tester cette opération.

Question 10 : Surchargez les opérateurs << et >> pour permettre l'affichage et la saisie d'un vecteur. Modifiez votre programme principal afin de tester ces opérateurs.

Question 11 : Les méthodes set et get de la classe Vecteur ne sont pas pratiques à utiliser. Surchargez l'opérateur [ ] (sous la forme de méthodes de la classe Vecteur) afin que ce dernier permettre d'*accéder* aux coordonnées du vecteur. Cet *accès* devra permettre :

- soit uniquement la consultation d'une coordonnée;
- soit aussi la modification d'une coordonnée.

Modifiez votre programme principal afin de tester cette nouvelle fonctionnalité.

Test unitaire

Pour tester vos programmes C++, vous utiliserez googletest. Voir le fichier ALIRE dans l'archive test_vecteur.tar.