Feuille TP n° 2 : Opérateurs

Chaque TP est divisé en deux phases qui doivent, dans la mesure du possible, être traitées simultanément. La première phase consiste en l'implémentation pure de la classe et de ses méthodes. La seconde phase s'attachera à l'analyse et à l'amélioration du code.

L'objectif de ce second TP est de travailler sur la surcharge d'opérateurs pour la classe `Dvector`. Pour ce faire, vous reprendrez comme base de travail la classe implémentée lors du premier TP.

Phase 1 : Implémentation

1. Implémenter l'opérateur d'accession `( )` à un élément du vecteur. Cette opérateur devra permettre un accès en lecture et en écriture lorsque cela a un sens.
2. Implémentation des opérateurs standards :
   - Implémenter l'addition, la soustraction, la multiplication et la division par un réel.
   - Implémenter l'addition et la soustraction de 2 vecteurs.
   - Implémenter l'opérateur unaire $-$.
3. Implémenter un constructeur permettant de créer un vecteur à partir d'un fichier dans lequel les éléments du vecteur sont stockés en colonne.
4. Implémenter les opérateurs << et >>.
5. Implémenter les opérateurs $+=, -=, *=, /=$ lorsque l'opérande droit est un réel ou un vecteur si cela a un sens.
6. Surcharger l'opérateur d'affectation $=$. On proposera au moins deux implémentations différentes dont l'une d'elle utilisera la fonction `memcpy`.
7. Surcharger l'opérateur de test d'égalité $==$.
8. Implémenter une méthode `view (bool copy, int i1, int i2)` permettant de manipuler (visualiser) une sous-partie d'un vecteur (un bloc) comme un vecteur à part entière. Cette fonction `view` créera donc un vecteur et prendra en argument 2 entiers i_1 et i_2 qui seront les 2 indices délimitant le bloc à considérer. Dans un souci d'efficacité, la fonction `view` ne procédera pas à une recopie des données sauf si `copy==true`. Pour ce faire, ajoutez à la classe `Dvector` un attribut permettant de savoir si l'instance de la classe est propriétaire ou non de son pointeur de données et modifiez en conséquence l'ensemble des méthodes précédemment écrites.

   Dans le cas de l'opération = entre les vecteurs `a` et `b`, on distingue les cas suivants :

   Si `a` et `b` ont la même taille

a	b	a=b
P	P	P
P	NP	P
NP	P	NP
NP	NP	NP
0	P	P
0	NP	NP

   Si `a` et `b` ont des tailles différentes

a	b	a=b
P	P	P
P	NP	P
NP	P	Err
NP	NP	Err
0	P	P
0	NP	NP

où P signifie que le vecteur est propriétaire de ses données, NP que le vecteur n'est pas propriétaire de ses données, Err signifie que l'opération doit retourner une erreur et 0 signifie que le vecteur n'a pas été initialisé. La troisième colonne de chacune des tables donne l'état de la variable `a` après l'opération `=`.

Phase 2 : Analyse

Question 1. *Pour chacune des méthodes précédentes, écrire un programme test.*

Question 2. *Mettre en évidence la différence entre l'utilisation des 2 opérateurs $+$ suivants :*

```
Dvector operator+ (Dvector a, Dvector b)
Dvector operator+ (const Dvector &a, const Dvector &b)
```

Comparer le nombre d'objets créés dans les 2 *cas.*

Question 3. *Comparer les performances de l'opérateur $=$ si on utilise ou non la fonction* `memcpy` *pour recopier les données. Tester l'efficacité de ces 2 versions sur des vecteurs de plusieurs millions d'éléments à l'aide de la fonction* `time`*. En particulier, on mesurera le temps nécessaire à la recopie sans utiliser* `memcpy` *et lorsque l'opérateur* `( )` *vérifie le non dépassement des bornes.*

Question 4. *Utiliser valgrind pour vérifier que toute la mémoire a été libérée.*