Introduction et Outils

Objectif

L'objectif de ce premier TP est de mettre en place les objets qui seront nécessaires à la réalisation des prochains TP, ainsi que l'utilisation de différents outils de programmation.

Dans un premier temps, vous implémenterez une classe vecteur dont l'élément de base sera le double, cette classe sera nommée `Dvector`. Cette classe devra utiliser l'allocation dynamique pour le stockage des données. Évidemment, vous n'utiliserez aucun conteneur STL pour votre implémentation.

Phase 1 : Implémentation

1. * Implémentation de constructeurs, du destructeur, et d'une méthode d'affichage :
   - Implémenter le constructeur par défaut.
   - Implémenter un constructeur prenant en argument la taille du vecteur à créer et un second argument optionnel qui spécifie la valeur à utiliser pour initialiser les éléments du vecteur.
   - Implémenter le destructeur.
   - Implémenter une méthode `void display ( std::ostream& str )` pour afficher un `Dvector`. Cette méthode affichera, **sur le flux passé en paramètre**, le contenu du vecteur (un élément par ligne). Par exemple `Dvector (3, 2).display ( std::cout )` enverra : `2\n 2\n 2\n` vers la console (au minimum deux décimales doivent être affichées).

Une fois la question 1 traitée, vérifier que la structure (dossiers, fichiers, nom des classes et fonctions) répond bien aux instructions données. Lancer le script `verifier.py`. Celui-ci doit confirmer que la question a été traitée correctement.

Relancer `verifier.py` régulièrement (pour chaque question), afin de vérifier que vous respectez bien l'énoncé. **Enfin, le relancer sur l'archive compressée (.tar.gz) avant de l'envoyer.**

2. Implémentation du constructeur par recopie, des méthodes d'accès et de manipulations de base :
   - Implémenter une méthode `int size ()` renvoyant la taille du vecteur.
   - Implémenter une méthode `void fillRandomly ()` qui remplit le vecteur selon la loi uniforme sur $[0, 1]$. On pourra dans un premier temps utiliser la fonction `random` de la librairie C.
   - Implémenter un constructeur par recopie.

   Pour l'instant, on ne demande pas de surcharger l'opérateur $=$, par conséquent on prêtera la plus grande attention à la différence qui existe entre les 2 écritures :

   ```
   Dvector x; x = Dvector (3, 1.); Dvector x = Dvector (3, 1.);
   ```

3. Implémenter un constructeur permettant de créer un vecteur à partir d'un fichier dans lequel les éléments du vecteur ont été stockés. Le constructeur prendra en paramètre le nom du fichier `Dvector ( std::string )`.
   - Un fichier créé avec la méthode `display` de la question 1, devra pouvoir être relu par ce constructeur. Cependant, le constructeur pourra être plus permissif (en acceptant des espaces et des lignes vides par exemple).
   - Si le fichier n'existe pas, un vecteur de taille nulle sera créé.
   - Afin de réaliser la lecture, utiliser les flux.

Phase 2 : Analyse

Question 1

Précisez dans votre rapport la différence entre les deux écritures :

```
Dvector x; x = Dvector (3, 1.); Dvector x = Dvector (3, 1.);
```

Question 2

Pour chacune des méthodes précédentes, écrire un programme de test. Un programme de test a pour but de vérifier le comportement de code de la façon la plus automatisée possible :

Ceci n'est pas un programme de test

```
Dvector v(3,2.5);
std::cout << v.size() << ":";
v.display( std::cout );
```

Ceci est un programme de test

```
Dvector v(3,2.5);
assert( v.size() == 3 );
std::stringstream str;
v.display( str );
assert( str.str() == "2.5\n2.5\n2.5\n" );
std::cout << "OK";
```

Question 3

Ajouter aux constructeurs une instruction d'affichage pour repérer les appels implicites aux constructeurs et faire de même avec le destructeur.

Question 4

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