TP n° 2 de C++ - Forme canonique de Coplien

Dans ce TP vous devez implémenter une classe chaîne de caractères et étudier les problèmes vus en TD et signalés dans l'ouvrage « Effective C++ » de Scott Meyers.

String
- _size : entier - _pData : tableau de char
+ String() + String(char * inCString) + String(String inString) +~String() + opérateur d'affectation + opérateur d'extraction de caractère + opérateur d'affichage

1) Constructeur de base et destructeur

- Ecrire une classe String qui a deux attributs : un entier _size et un pointeur sur un caractère _pData. (_size est la taille du tableau _pData). Le schéma UML ci-dessus précise avec les '-' que les attributs sont privés et que les méthodes avec '+' sont publiques.
- Ecrire un constructeur par défaut sans arguments qui initialise les attributs à 0 et (void *) 0 pour le pointeur (les nouveaux codes C++ vont bientôt utiliser **ptrnull**.
- Ecrire un constructeur qui prend en paramètre une chaine de type langage C (char *) inCString, qui alloue correctement _pData et qui copie inCString dans _pData. La fonction strcpy() se trouve dans le fichier d'entête <cstring>.
- Ecrire une fonction main() qui instancie un objet s1 de type String avec une valeur pour tester le constructeur avec argument.
- Vérifier avec *valgrind* qu'il y a bien une fuite mémoire
- Ajouter à la classe String un destructeur qui rend la mémoire allouée.
- Vérifier avec le logiciel *valgrind* que vous n'avez plus de problèmes.

2) Constructeur de copie

- Modifier le programme principal pour qu'il ne déclare tout d'abord qu'une chaine s1 de taille 10 puis qui déclare et initialise une chaine s2 à partir de s1 sur la même ligne (String s2 = s1). Quel sera le problème ? Vérifier avec *Valgrind*.
- Corriger le problème en proposant un constructeur de copie et prévoir une ligne dans le constructeur de copie pour afficher sur la sortie standard "Constructeur de copie appelé"
- Ecrire une méthode toScreen() qui affiche la chaine sur la sortie standard (std ::cout)
- Ecrire une **fonction** displayByValue() qui prend une instance de String en paramètre et qui appelle la méthode afficher() de la chaine en paramètres.

- Ecrire une autre **fonction** displayByReference() qui prend une référence sur une chaine et qui fait la même chose.
- Tester l'appel de ces fonctions et vérifier que le constructeur de copie est bien appelé lors d'un passage de paramètre par valeur.

3) Surcharge de l'opérateur d'affectation

- Proposer maintenant un constructeur qui accepte en entrée une taille – inSize et qui effectue l'initialisation des attributs (allocation dynamique à la bonne taille pour _pData). Dans le programme principal, créer 2 instances s1 et s2 de taille 10 et 20 puis affecter s2 à s1.
- Exécuter le programme, noter le message d'erreur puis vérifier avec *valgrind*.
- Corriger l'erreur en proposant un opérateur d'affectation, vérifier avec *valgrind*. L'opérateur prend en paramètre une référence sur un objet constant et renvoie une référence ("*this") pour le chainage d'affectations s1 = s2 = s3.
- Bien vérifier que le cas s1 = s1 ne puisse provoquer d'erreur

4) Surcharge d'opérateurs(<<, [] et +)

En utilisant les éléments de syntaxe du polycopié :

- Proposer une surcharge de l'opérateur <<. Cet opérateur (une fonction) prend deux références en paramètres : un flux et une chaine et renvoie le flux pour rendre le chainage possible. Vérifier par un std::cout << myString; ce qu'il se passe réellement (la référence sur String doit être constante – au besoin modifier et compiler). Vous aurez peut être besoin d'ajouter un accesseur sur _pTab ou alors d'utiliser le principe des fonctions amies.
- Ajouter un opérateur [] permettant la modification d'un élément de la chaîne.
  Tester l'implémentation sur une de vos instances en modifiant la valeur d'un caractère.
- Essayer de modifier l'implémentation de l'opérateur << pour qu'il affiche chaque caractère de la chaîne ligne à ligne à l'aide de l'opérateur précédemment défini.
  Ex : Pour la chaîne « Toto », on aura l'affichage suivant :
  « T
  o
  t
  o »
  Noter le message d'erreur à la compilation.
- Pour palier cette erreur, ajouter un nouvel opérateur [] constant (méthode constante) pour accéder à un caractère de la chaine en lecture seule.
- Ajouter un opérateur de concaténation de chaînes en utilisant le symbole + de manière naturelle.

Notes :

- ✓ Utiliser le mot clé const sur toutes les méthodes qui ne changent pas l'état de l'objet courant.
- ✓ std::cout est un objet de type std::ostream (défini dans l'entête <ostream>)

Pour ceux qui s'ennuient :

DoubleVector
- _size : entier - _pData : tableau de Double
+ DoubleVector() + DoubleVector (int inSize) + DoubleVector(double * inDArray) + DoubleVector(DoubleVector & inDVector) +~DoubleVector() + opérateur d'affectation + opérateurs d'indexation [] + opérateur d'affichage avec flux << + opérateur d'addition

1) Ecrire le code C++ d'une classe DoubleVector de réels double précision sur le modèle de la classe String. Identifier les attributs et les méthodes de base pour une telle classe. Notamment, pour ce vecteur on prévoit l'écriture d'un constructeur de copie, d'un opérateur d'affectation, d'un opérateur d'addition, d'un opérateur d'indexation et de flux. Réaliser l'implémentation dans deux fichiers : un fichier entête et un fichier d'implémentation. Ecrire le corps de la fonction main dans un fichier séparé. Tester l'option –MM du compilateur g++ (g++ -MM *.cpp) puis proposer un makefile. Tester dans le programme principal les différentes méthodes du DoubleVector une à une. Prenez le temps de les développer et de les tester au fur et à mesure.