TP 2

1 Anatomie d'une classe

Finir avec soin cet exercice du TP1. Il est recommandé d'ajouter l'affichage d'une ligne vide entre chaque création d'objet dans le *main*, pour bien voir quelles opérations sont appelées de manière automatique. N'oubliez pas d'écrire vos réponses (sur une feuille ou en commentaire dans votre fichier source).

2 Capsules RAII, construction et affectation par déplacement

Une classe correspond à une capsule RAII (Resource Acquisition Is Initialization) si les instances de cette classe deviennent propriétaires/responsables de ressources allouées dans le tas (par exemple au moment de leur initialisation par un constructeur). La désallocation de ces ressources devra alors précéder la disparition de l'instance (rôle du destructeur). Modifier la classe LaClasse de manlère à en faire une capsule RAII (faire un dessin pour montrer que vous avez bien compris le concept) et doter la classe LaClasse d'un constructeur par déplacement ainsi que d'un opérateur d'affectation par déplacement.

3 Constructeurs, destructeurs et affectation des classes dérivées

Dérivez la classe *LaClasse* de l'exercice précédent en *LaClasseSpecialisee* et munir la classe obtenue de constructeurs, destructeur et opérateur d'affectation laissant une trace d'affichage. Regardez ce qu'il se passe :

- quand vous omettez de définir des constructeurs dans *LaClasseSpecialisee*,
- quand vous omettez de définir un constructeur par copie,
- puis quand vous oubliez de recourir aux listes d'initialisation dans la définition des constructeurs (si jamais vous en avez absolument besoin, n'hésitez pas à passer en *protected* les données membres de la classe de base).

Est-ce bien cohérent avec ce que vous avez vu en cours ?

Amusez-vous à tester les possibilités d'`upcast` et `downcast` de pointeurs et de références au sein de votre hiérarchie de classe *LaClasse*.

4 Un module *String* fait main

Ecrivez une classe String permettant la manipulation de chaînes de caractères.

Dans cet exercice, vous pourrez vous servir de la classe *LaClasse* comme rappel de syntaxe des éléments qu'on peut trouver dans une classe. Assurez-vous de votre gestion saine de la mémoire avec `valgrind`. Bien entendu le but n'est pas que vous recouriez à un conteneur de la STL, mais que vous gériez vous même l'allocation dynamique de la mémoire nécessaire au stockage. Comme toujours, vous enchaînerez des cycles de :

1. développement d'une nouvelle fonctionnalité ;
2. compilation ;
3. exécution et vérification de son bon fonctionnement.

Vous ne passerez à une nouvelle opération que lorsque la précédente sera validée dans l'ensemble des cas de figures qui peuvent être rencontrés.

Votre classe String devra satisfaire (notamment) aux contraintes suivantes :

— Possibilité de construction à partir d'une chaîne de caractères ordinaire (ou chaîne vide par défaut) ;
— Possibilité de considérer un caractère comme une chaîne d'un seul caractère (constructeur à partir d'un *char*) ;
— Possibilité de conversion en une chaîne de caractères ordinaire, pour pouvoir utiliser certaines des fonctionnalités de la bibliothèque standard C-ANSI qui ne modifient pas la chaîne (Attention toutefois à bien protéger votre classe String !)
— Opérations de concaténation (+) et de comparaison (==, !=, <, >, <=, >=) entre chaînes ;
— Accès en lecture (et en écriture si c'est possible) à la $i$ème lettre d'une chaîne (opérateurs []) ;
— Accès à la longueur d'une chaîne ;
— Test d'une chaîne vide ;
— Fabrication d'une sous-chaîne (à partir de 2 indices début et fin) ;

On munira également la classe d'opérations d'entrée/sortie (utilisation des classes *ostream* et *istream* avec surcharges des opérateurs « et »).

5 Comparatif Java et C++

Aviez-vous fini cet exercice ? Quelles sont vos conclusions en terme d'efficacité ? Essayez de comprendre ce qu'il se passe en mémoire en réalisant un dessin de l'occupation mémoire d'une Image en JAVA et en C++ respectivement.