Travaux Dirigés Programmation C++ : Feuille 3

Informatique 2ème année.

—*Julien Allali* - `allali@enseirb.fr` —

Polymorphisme

▸**Exercice 1.** *Polymorphisme choisi ? (Correction)*
Récupérer le fichier `Exemple.cpp`

Étudier le code, écrire **sur une feuille** *le diagramme de classe, compiler et exécuter.*
Pour chacune des modifications suivantes, recompiler et exécuter le programme :
— *ajouter le mot clé* `virtual` *uniquement devant* `Mere::message`
— *ajouter le mot clé* `virtual` *uniquement devant* `Fille::message`
— *ajouter le mot clé* `virtual` *devant* `Mere::message` *et* `Fille::message`.
Trouver le sens de ce mot clé (cela a un rapport avec le titre de l'exercice).

*Essayer de reproduire cette exemple en utilisant l'***allocation automatique**. *Pouvez-vous en tirer une conclusion sur l'utilisation du polymorphisme en C++ ?*

Utiliser la fonction `main` *commentée en bas du fichier et re-tester avec ou sans* `virtual` *sur* `Mere::message`. *Expliquer le résultat observé.*

▸**Exercice 2.** *Polymorphisme et référence*
Reprendre l'exercice précédent et utiliser le `main` *suivant :*

```
int main(){
  Fille *f=new Fille();

  Mere &m=*f;

  m.message();

  delete f;
}
```

Indiquer les allocations, instanciations (appels aux constructeurs), libérations et destructions successives.

Remplacer la dernière instruction par `delete &m;`. *Le code est-il toujours valide ? Justifier.*

Supprimer tout les `virtual` *puis ajouter un destructeur dans chacune des classes et vérifier que le bon destructeur est appelé.*

Règles de programmation : Toujours mettre le destructeur des classes potentiellement dérivables en virtuel.

Forme Canonique

Pour assurer une gestion mémoire convenable d'une classe possédant des allocations dynamiques, il faut la définir avec une forme minimale « canonique » (Coplien) : constructeur, destructeur, constructeur par recopie et opérateur d'affectation. La classe `Chaine` [(1)] développée précédemment respecte cette forme canonique.

Vous trouverez le source de cette classe dans le fichier archive `chaine3.tar`.

▸**Exercice 3.** *Une classe classique . . .*

(1). La bibliothèque STL fournit déjà une classe `string` pour la gestion des chaînes de caractères avec le fichier d'en-tête `<string>`. Mais alors où était le challenge ! !

Dans deux fichiers `Personne.hpp` *et* `Personne.cpp`*, écrire le code d'une classe* `Personne` *qui fait partie de l'espace de nom* `enseirb`*. Cette classe a un attribut privé* `_nom` *de type* `Chaine`*.*

Dans la classe `Personne`*, écrire un unique constructeur qui prend une instance de* `Chaine` *en argument. Écrire également une méthode publique* `nom()` *retournant la valeur de l'attribut.*

Dans le programme principale, écrire la fonction `void afficheNom(Personne& p)` *qui affiche sur la sortie standard le nom de la personne* [(2)]*.*

Sans écrire la forme canonique d'une classe, vérifier que tout se passe bien au niveau de la mémoire.

Héritage

Dans les exercices qui suivent, veillez à bien séparer chaque classe dans différents fichiers (comme en Java). Écrivez également un `Makefile` pour faciliter la compilation. L'ensemble des classes ci-dessous font partie de l'espace de nom `enseirb`. Après chaque classe écrite, vous testerez votre code à l'aide d'un fichier d'exemple.

▸**Exercice 4.** *étudiants . . .*
Écrire une classe `Etudiant` *dérivée de* `Personne`*. Cette classe dispose d'un attribut* `_filiere` *de type* `Chaine` *(par exemple :* **"Info1"**, **"Info2"**, *. . .) et d'un attribut* `_enseignement` *(par exemple :* **"algo1,prog1"** *ou* **"poo, c++, prog. sys"***).*

De plus, elle a trois méthodes : `filiere()` *(retournant la valeur de l'attribut),* `enseignement(...)` *(retournant la valeur de l'attribut),* `setEnseignement(...)` *(positionnant la valeur de l'attribut).*

Redéfinir la méthode `nom()` *afin que celle-ci renvoie le nom précédé de la mention* **"Eleve:"**.

Appliquer la fonction `afficheNom()` *sur une instance de cette classe.*

Comme écrire la méthode `nom()` *si son prototype renvoie une référence constante.*

▸**Exercice 5.** *enseignant. . .*
Écrire la classe `Enseignant`*, sous-classe de* `Personne`*. Cette classe dispose d'un attribut* `_service` *de type* `int` *indiquant le nombre d'heures d'enseignement effectuées et une méthode* `nbHeure()` *retournant la valeur de cet attribut.*

Redéfinir la méthode `nom` *afin que celle-ci renvoie le nom précédé de la mention* **"Enseignant:"**.

▸**Exercice 6.** *élève vacataire*
Certains élèves peuvent effectuer des enseignements (vacations). En utilisant l'héritage multiple, Écrire une classe `EleveVacataire` *héritant de* `Enseignant` *et* `Eleve`*.*

La redéfinition de la méthode `nom()` *appel la méthode* `nom()` *des deux classes de base.*

Appliquer la fonction `afficheNom()` *sur une instance de cette classe.*

Expliquer le problème rencontré et la manière de le résoudre.

Partage de données

La classe `Chaine` définit des objets constants, il est donc possible d'effectuer un partage de la zone mémoire contenant les caractères (plutôt qu'une recopie de la zone mémoire).

Pour ce partage, nous avons besoin d'un mécanisme de compteur de références.

Vous trouverez dans l'archive `smartchar.tar` une réalisation d'un `SmartPointer` pour la gestion du partage d'un zone mémoire de type `char`.

Un `SmartPointer` est une classe dont les instances se comportent comme des pointeurs [(3)]. Cette classe conserve une zone mémoire et gère sa libération en associant un compteur à cette zone. Ce `SmartPointer` n'effectue pas l'allocation de la zone mémoire concernée. Il donne accès à cette zone par les opérateurs `*`, `->`, l'opérateur de conversion et la méthode `get()`.

La réalisation du `SmartPointer` respecte la forme canonique.

(2). Vous pouvez utiliser les flots d'entrée/sortie pour afficher ce nom mais il faut surcharger `operator<<` pour la classe `Chaine`.

Dans la classe vous trouverez deux attributs : `data` qui pointe vers la donnée prise en charge par le `SmartPointer` et `counter` qui compte le nombre de `SmartPointer` pointant vers cette même donnée.

▸**Exercice 7.**

Dans la classe `Chaine` *utiliser les* `SmartPointer` *pour l'attribut* `_donnees`.

Dans cette exercice, la concaténation (`operator+`*) alloue toujours une nouvelle zone mémoire pour contenir la valeur de la concaténation. Il y a duplication du contenu de deux zones mémoires (c'est le thème de l'exercice suivant).*

Effectuer l'ensemble des modifications nécessaires et supprimer le code superflu.

▸**Exercice 8.** *Concaténation avec partage.*

Pour effectuer une concaténation avec partage de donnée, une solution est de stocker une séquence d'instances de `SmartPointer` *dans la classe* `Chaine`.

Mettre en œuvre cette solution en utilisant la classe `std::vector` *de la bibliothèque STL (fichier en-tête* `<vector>`*).*

(3). `C++11` inclut la classe `std::shared_ptr` qui est une réalisation d'un ≪ smartpointer ≫ avec compteur référence.