Programmation Générique
M1 Informatique — Examen 14 avril 2008
2 heures

Les documents, hors livres, sont autorisés. Les **5** exercices sont indépendants.

Exercice 1 :
La bibliothèque BOOST propose une classe `noncopyable`. Cette classe est telle que tout objet d'une classe qui hérite de `noncopyable` ne peut être dupliqué, ou servir d'argument gauche d'un opérateur d'affectation. Écrire une implémentation de la classe `noncopyable`.

Exercice 2 : On veut écrire une classe `polynomial` permettant de manipuler des polynômes dont les coefficients sont de type paramétrable `T`.

1. Cette classe a deux constructeurs (en plus du constructeur par copie), un sans argument qui initialise le polynôme à zéro, l'autre qui prend un `std::vector` et qui crée un polynôme tel que le coefficient de X^i est donné par le coefficient i du vecteur.
2. La classe comporte une méthode `degree` qui retourne le degré du polynôme.
3. Les opérateurs arithmétiques (`+`, `-`, `*`), et d'affectation (`=`, `+=`, `-=`, `*=`) sont surchargés pour cette classe, ainsi que le test d'égalité et de non égalité.
4. L'opérateur `[]` permet d'accéder à chaque coefficient, aussi bien en lecture qu'en écriture. Une tentative d'accès en dehors de l'intervalle [0 ;degré] provoquera une exception qu'on définira.
5. L'opérateur `()` permet de passer une valeur de type `T` au polynôme et retourne la valeur du polynôme en ce point.

Écrire la classe `polynomial`. On veut par exemple que le code suivant fonctionne :

```
const polynomial<double> p(new std::vector<double>(4,1.0));
poynomial<double> q(p*p);
q+=p;
q[4]=5.2;
std::cout << q(3.7) << std::endl;
```

Exercice 3 :
On désire écrire un module permettant de gérer des logins et mots de passe. Ce module offre une classe `login_t`, qui contient :

- un constructeur prenant en arguments un login et un mot de passe ;
- une méthode `get_login` permettant de récupérer un mot de passe
- une méthode `chg_passwd` permettant de modifier le mot de passe associé à ce login,
- une méthode statique `get_hash` qui retourne un hash code `long long int` de son argument,
- une méthode `get_passwd_hash` retournant un `long long int` correspondant au hash code du mot de passe de l'objet.

On ne veut pas que les utilisateurs de la classe connaissent les implementations de ces méthodes, ni même le nom et le type sous lesquels un objet `login_t` stocke le login et le mot de passe. On suppose que pour chaque plateforme, on fournit aux utilisateurs un fichier `login.hh` et un fichier `login.o`.

Écrire le fichier `login.hh` qui déclare la classe `login_t`.

Exercice 4 : Le pgcd de deux nombres positifs et défini par la récurrence suivante :

$$\mathsf{pgcd}(n,m) = \begin{cases} m & \text{si } n = 0; \\ \mathsf{pgcd}(m \bmod n, n) & \text{si } 0 < n \leqslant m; \\ \mathsf{pgcd}(m,n) & \text{sinon.} \end{cases}$$

Écrire les classes nécessaires pour que, étant données deux constantes `N` et `M`, l'expression `gcd<N,M>()::val` soit une constante égale au pgcd de `N` et `M`.

Exercice 5 : On considère le code suivant :

```
#include<iostream>

struct classe_A{
   virtual void ma_fonction() = 0;
};

struct classe_B : classe_A{
   void ma_fonction(){
     std::cout << "classe B" << std::endl;
   }
};

int main(){
  classe_B b;
  classe_A& a=b;
  a.ma_fonction();
}
```

1. Comment qualifie-t-on la déclaration de `ma_fonction` dans `classe_A` ; comment qualifie-t-on `classe_A` elle-même ?
2. Quel est le comportement de ce programme ; comment fonctionne l'appel `a.ma_fonction()` à l'exécution ?
3. On veut éliminer l'usage de `virtual` afin d'écrire :

   ```
   int main(){
     classe_B b;
     classe_A<classe_B>& a=b;
     a.ma_fonction();
   }
   ```

   Écrire les nouvelles versions de `classe_A` et `classe_B` afin de garder le même comportement.
4. En maximum dix lignes, expliquer quels sont les avantages de chacune des implémentations.