Université Paris Diderot
Master 1 II

Langages à Objets Avancés
Année 2012–2013

Examen

Jeudi 16 Janvier 2013

Tous documents autorisés (sauf copie du voisin, appareillage électronique, etc). Les téléphones portables, comme tout autre moyen de communication vers l'extérieur, doivent être éteints. Le temps à disposition est de trois heures. Motivez bien vos réponses.

1 Exercice

1. Indiquer (en justifiant) ce que ce programme affiche lors de son exécution.
2. À quoi sert le mot-clé `virtual` devant la définition des destructeurs ? Donner un exemple permettant d'illustrer sa nécessité.

```
#include <iostream>
using namespace std;
class Premiere {
public:
  Premiere()             { cout << "ctor premiere" << endl; }
  virtual ~Premiere()    { cout << "dtor premiere" << endl; }
  void salut()           { cout << "salut" << endl; }
  virtual void bonjour() { cout << "bonjour" << endl; }
};
class Seconde : public Premiere {
public:
  Seconde()              { cout << "ctor seconde" << endl; }
  virtual ~Seconde()     { cout << "dtor seconde" << endl; }
  void salut()           { cout << "salutations" << endl; }
  virtual void bonjour() { cout << "bien le bonjour" << endl; }
};
int main() {
  Premiere p;
  Seconde *s = new Seconde;
  p.salut(); p.bonjour();
  (*s).salut(); (*s).bonjour();
  Seconde *ps = s;
  ps->salut(); ps->bonjour();
  Premiere *pp = ps;
  pp->salut(); pp->bonjour();
  delete s;
  return 0;
}
```

2 Exercice

Soit l'extrait de programme suivant :

```
void quickSort(int arr[], int left, int right) {
      int i = left, j = right; int tmp; int pivot = arr[(left + right) / 2];

      /* partition */
      while (i <= j) {
            while (arr[i] < pivot) i++;
            while (arr[j] > pivot) j--;
            if (i <= j) {
                  tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
                  i++;
                  j--;
            }
      }
      /* recursion */
      if (left < j) quickSort(arr, left, j);
      if (i < right) quickSort(arr, i, right);
}
```

On souhaite (pour certaines données expérimentales) compter le nombre de certaines classes d'opérations effectuées durant l'exécution de la fonction `quickSort` (mesure expérimentale de la complexité algorithmique).

1. Transformer/adapter le code donné de sorte que l'on puisse compter diverses opérations (par exemple les tests et/ou les affectations). Attention : car la seule modification acceptée dans le code précédent est le remplacement de certaines types ; aucune instruction supplémentaire ne doit être retirée ou ajoutée. Bien entendu, il est peut-être nécessaire de construire par ailleurs des types particuliers (si c'est le cas, il faudra en fournir le code).

3 Exercice

1. Écrire un type `List<`T`>` permettant de représenter une structure d'éléments (du même type) et chaînés (à la construction d'un élément `List<`T`>`, la liste sera vide, et l'on disposera de 4 méthodes : `add(`T`&)`, T`& remove(`i`)`, `bool isIn(`T`&)` et T `get(`i`)` permettant respectivement d'ajouter un élément en fin de liste, de supprimer un élément de la liste étant donné sa position dans la liste, de tester si un élément est dans la liste et de récupérer l'élément d'indice donné.
2. Puisqu'il s'agit d'un modèle de classe, préciser les contraintes imposées sur le type pour que l'instanciation du modèle fonctionne.
3. Écrire un itérateur de cette structure, un tel itérateur devra pouvoir être obtenu par une instruction du genre `l::iterator i = l.getIterator();` où `l` est une liste instanciée. Les méthodes disponibles sur un itérateur sont `getCurrent` et `next` qui, respectivement, renvoient l'élément courant de l'itération en cours et déplace le « curseur » vers l'élément suivant (s'il y en a un) et renvoi un booléen pour indiquer qu'il y a bien un élément courant.

4 Problème

La phylogénie nous indique que les Bovins (qui ont un cuir d'une certaine épaisseur) et les Ovins (qui sont couverts de laine d'une certaine longueur) sont des Ruminants (qui ruminent) eux-même des Mammifères (qui allaitent leurs petits). Parmi les Bovins on trouve les Bœufs dont la représentante la plus connue est Marguerite, jolie vache laitière de 564kg et élevée par Marcel et Germaine Delapeau, fermiers à Brive-la-gaillarde. Pour information, les Vaches sont des Bœufs femelles.

1. Comment traduit-on en UML une telle spécification ?
2. Placer dans les classes identifiées les caractéristiques/opérations sous-entendues par la spécification.
3. Traduire chaque classe UML en une classe C++ correspondante avec attributs et méthodes. Prendre soin de spécifier tous les modificateurs adéquats, nécessaires ou utiles (`const`, etc).
4. Écrire un programme C++ permettant de créer une version numérique de la charmante Marguerite et de ses propriétaires et d'afficher leurs caractéristiques.
5. Écrire une usine (factory) permettant de créer un Bœuf de sexe, nom et race quelconques (sous la forme d'attributs). Modifier la classe Vache en conséquence. Modifier la construction de Marguerite afin d'obtenir une Vache de race Limousine.
6. Si l'on souhaite obliger l'utilisateur à passer par l'usine pour obtenir une Vache, que doit-on modifier ?
7. Si l'on décide de distinguer les races de Bœufs à l'aide de classes différentes et non plus d'attributs, qu'est-ce que cela change ? Écrire les classes Limousine et Normande.
8. Sachant qu'une Limousine fait « Meuheuh » et une Normande fait « Meeeuuuhhh » juste après avoir mangé, implanter ce mécanisme dans les classes adéquates. Attention, les Ovins eux, ne font aucun bruit particulier après avoir mangé...