Examen (Sujet A)
Programmation orientée objet avancée

— *Master d'informatique* —

Janvier 2016, durée 2h.

L'examen se compose de quatre exercices indépendants. Les seuls documents autorisés sont une feuille de mémento (double A4). Tous les appareils électroniques sont interdits à l'exception des montres (et encore).

Les réponses aux questions doivent directement être écrites dans les cadres du sujet. Tout ce qui est écrit en dehors des cadres n'est pas pris en compte pour l'évaluation.

▶ **Exercice 1** Donner les affichages du programme suivant.

```
class B {
public:
  string f() const { return g() + " by f() in B"; }
  virtual string g() const { return "g() in B"; }
};

class D : public B {
public:
  string f() const { return g() + " by f() in D"; }
  virtual string g() const { return "g() in D"; }
};

int main() {
  cout << B().f() << endl;
  cout << B().g() << endl;
  cout << D().f() << endl;
  cout << D().g() << endl;
  cout << ((B) D()).f() << endl;
}
```

▶ **Exercice 2** On désire modéliser un système de fichiers contenant uniquement des fichiers et des répertoires. La spécification est la suivante.

+ chaque fichier/répertoire a un identifiant unique (entier)
+ un fichier a une longueur (entier)
+ un répertoire contient des associations entre des noms (string) et des fichiers/répertoires
+ chaque fichier/répertoire a une méthode `size()` qui retourne sa *taille*, c'est-à-dire
  - sa longueur pour un fichier
  - son nombre d'asssociations pour un répertoire
+ chaque fichier/répertoire a une méthode `disk()` qui retourne son *occupation disque*, c'est-à-dire
  - sa longueur pour un fichier
  - la somme de sa taille multipliée par 32 et des occupations des fichiers/répertoires qu'il référence.

a) Écrire des classes `File` et `Directory` pour les fichiers et les répertoires. Le code suivant doit fonctionner avec les classes écrites.

```
int main() {
  File f1(100);                 // Création d'un fichier f1 de taille 100
  cout << f1.size() << endl;    // 100
  cout << f1.disk() << endl;    // 100
  Directory d1;                 // Création d'un répertoire vide d1
  cout << d1.size() << endl;    // 0
  cout << d1.disk() << endl;    // 0
  d1.add("f1", &f1);            // Ajout de f1 à d1 sous le nom f1
  cout << d1 << endl;           // Dir 1: f1 --> 0 (100)
  File f2(1000);                // Création d'un fichier f2 de taille 1000
  d1.add("f1", &f2);            // Remplacement de f1 par f2 dans d1
  cout << d1 << endl;           // Dir 1: f1 --> 2 (1000)
  d1.add("f2", &f1);            // Ajout de f1 à d1 sous le nom f2
  cout << d1 << endl;           // Dir 1: f1 --> 2 (1000), f2 --> 0 (100)
  cout << d1.size() << endl;    // 2 (2 liens)
  cout << d1.disk() << endl;    // 1164 = (32 * 2 + 1000 + 100)
  Directory d2;                 // Création d'un répertoire vide d2
  d2.add("f3", &f1);            // Ajout de f1 à d2 sous le nom f3
  d2.add("d1", &d1);            // Ajout de d1 à d2 sous le nom d1
  cout << d2 << endl;           // Dir 3: d1 --> 1 (2), f3 --> 0 (100)
}
```

▶ **Exercice 3** Donner les `Vtable` des quatre classes suivantes. Pour chaque `Vtable`, donner les adresses qu'elle contient.

```
class B0 {                              |    class B1 {
public:                                 |    public:
  int f0() { return vb0; }              |      int f() { return vb1; }
  virtual int g0() { return vb0; }      |      virtual int g() { return vb1; }
protected:                              |    protected:
  int vb0 = 0;                          |      int vb1 = 1;
};                                      |    };
----------------------------------------+----------------------------------------
class D : public B1 {                   |    class C : public B0, public D {
public:                                 |    public:
  int f() { return -vd; }               |      virtual int g() { return vd + vb1; }
  virtual int g() { return -vd; }       |      virtual int h() { return vd + vb0; }
protected:                              |    };
  int vd = 2                            |
};                                      |
```

▶ **Exercice 4** On considère le programme suivant incomplet.

```
template<class Iterator, class Fun>
void map(Iterator begin, Iterator end, Fun function) {
  while (begin != end) {
    *begin = function(*begin);
    ++begin;
  }
}

class C {
  friend ostream& operator<<(ostream&, C);   // Friend output operator
public:
  C(int v) : value(v) {}                     // Constructor
  // Méthode à ajouter ICI
private:
  int value;                                 // Value
};
```



```
// Output operator for the class C
ostream& operator<<(ostream& out, C c) { out << c.value; return out; }

int main() {
  int t[4] = { 1, 2, 3, 4};                  // Array of 4 integers
  map(t, t+4, Times2<int>());                // Double each integer
  print(t, t+4);                             // Print: 2 4 6 8

  vector<string> v { "He", "l", "l", "o"};   // Array of 4 strings
  map(v.begin(), v.end(), Times2<string>()); // Double each string
  print(v.begin(), v.end());                 // Print: HeHe ll ll oo

  C r[4] = { C(1), C(2), C(3), C(4) };       // Array of 4 C instances
  map(r, r+4, Times2<C>());                  // Double each C object
  print(r, r+4);                             // Print: 2 4 6 8
}
```

Compléter le programme pour qu'il fonctionne correctement et affiche les commentaires sur chaque ligne de `print`.

a) Écrire la classe générique `Times2` telle que ces instanciations avec les types `int`, `string` et `C` fonctionnent avec la fonction générique `map`.
b) Écrire la fonction générique `print` qui affiche les valeurs avec un espace entre chaque valeur.
c) Compléter la classe `C` en ajoutant une méthode pour que la classe `Times2` fonctionne avec le type `C`.
d) Réécrire le premier appel à `map` en utilisant une lambda-expression.