Trimestre Hiver, 2006

Mohamed Lokbani

IFT1169 – Examen Final –

Inscrivez tout de suite votre nom et code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : mardi 18 avril 2006

Durée : 2 heures et 45 minutes (de 18h30 à 21h15)

Local : Z-110 ; Pavillon Claire-McNicoll (ancienne aile Z).

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises et clairement présentées.**

1._____/10 (1.1, 1.2)

2._____/15 (2.1, 2.2, 2.3)

3._____/20 (3.1, 3.2, 3.3)

4._____/25 (4.1)

5_____ /30 (5.1, 5.2, 5.3)

Total:_____/100

Directives officielles

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remettra sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes.

* Un seul étudiant à la fois peut quitter la salle.

* Toute infraction relative à une fraude, un plagiat ou un copiage est signalée par le surveillant au directeur de département ou au professeur qui suspend l'évaluation.

F.A.S

Exercice 1 (10 points)

1.1 Au niveau de l'utilisation, on remarque plusieurs différences entre le type « `char*` » et le type « `string` ». À l'aide de simples instructions de code, citer au moins 3 différences.

1.2 Soit la règle générale à respecter : « `Il ne faut jamais lever d'exceptions dans un destructeur` ». Expliquer le pourquoi de cette règle?

Exercice 2 (15 points) Soit le type énuméré « `ErrFic` » qui liste les différentes erreurs « `Entrée` » et « `Sortie` » sur des fichiers. Il est défini comme suit :

```
enum ErrFic {
   PasErrFic,  // Aucune erreur n'a été détectée
   ErrOuvLect, // Erreur lors de l'ouverture du fichier en lecture
   ErrOuvEcr,  // Erreur lors de l'ouverture du fichier en écriture
   FinFicSub,  // Atteint la fin du fichier à un moment inattendu
   ErrFicvide  // Le fichier est vide, ne contenant pas donc de données
};
```

En utilisant le type énuméré « `ErrFic` », écrire la définition des 3 fonctions suivantes :

2.1 `ErrFic OuvFicLec(ifstream& fic_entree, char *nom_fic);`
`// Ouvre le fichier « nom_fic » en lecture`
`// et retourne l'état de l'ouverture`

2.2 `ErrFic OuvFicEcr(ofstream& fic_sortie, char *nom_fic);`
`// Ouvre le fichier « nom_fic » en écriture`
`// et retourne l'état de l'ouverture`

2.3 `ErrFic CopieNCars(ifstream& fic_entree, ofstream& fic_sortie, int NbreCars);`

```
// La fonction vérifie que les deux fichiers sont ouverts,
// puis elle copie un nombre de caractères « NbreCars » du fichier
// « fic_entree » vers le fichier « fic_sortie ».
```

Exercice 3 (20 points) Soit la fonction générique « `moyenne` » définie comme suit :

```
template <class T> T moyenne(T tab[], int taille) {
   T total = 0;
   for (int i = 0; i < taille; i++)
   total = total + tab[i];
   return total / taille;
}
```

3.1 Cette fonction va fonctionner correctement pour :

a) N'importe quel type qui supporte l'addition.
b) N'importe quel type qui supporte la division par un entier.
c) N'importe quel type qui supporte la division par lui-même.
d) Les réponses a) et b).
e) Les réponses a) et c).

Cocher la bonne réponse et expliquer votre choix :

3.2 Soit la classe « `Valeur` » :

```
class Valeur {
private:
   int x;
public:
   Valeur(int i = 0) : x(i) {}
   Valeur(const Valeur& v) { x = v.x; }
   Valeur operator/(int d) { return Valeur(x / d); }
   bool operator==(const Valeur& Autre){
          return (this == &Autre);
   }
};
```

Est-ce que la fonction générique « `moyenne` » va fonctionner correctement dans le cas de la classe « `Valeur` »? [`Oui/Non` et justifier votre réponse. Dans le cas d'une réponse « `Non` », citer les changements à apporter au niveau de la classe « `Valeur` »].

3.3 Tout en détaillant votre démarche, que va afficher en sortie la méthode « `main` » suivante qui utilise la fonction « `moyenne` » décrite précédemment.

```
int main(){
   int a[] = {10, 20, 30, 40};
   int r = moyenne(a, 3);
   cout << "Moy = " << r << endl;
   r = moyenne(a, 4);
   cout << "Moy = " << r << endl;
   return 0;
}
```

Exercice 4 (25 points) Le programme suivant lit ligne par ligne à partir d'un fichier ouvert en lecture des données séparées par une virgule. Une lecture rapide de ce programme montre que le programmeur a certaines lacunes du coté optimisation et du coté conception orientée objet. D'une part, il a mal décomposé son programme, d'autre part l'abstraction de données est inexistante.
On vous demande de rebâtir ce programme en tenant compte des fondements de base qui sont derrière la programmation orientée objet.
Commencer d'abord par construire deux classes. La première classe va se charger de lire les données du fichier afin de les stocker. La seconde classe, elle va subdiviser chaque ligne de données en deux champs puis stockera ces derniers dans les variables membres appropriées.
Vous devez montrer clairement l'interface associée à chaque classe. Vous devez par la suite définir ces deux classes i.e. écrire le code complet des méthodes membres. Vous pouvez représenter les données avec d'autres structures de données plus efficaces. Et finalement, écrire la nouvelle méthode « `main` » qui va faire appel à ces deux classes. En résumé, vous devez:
- Écrire le code complet de la classe qui lit et stocke les données,
- Écrire le code complet de la classe qui manipule les données déjà lues,
- Écrire la nouvelle méthode « main ».

```
#include <string>
#include <iostream>
#include <fstream>
#include <list>
using namespace std;
int main(int argc, char **argv) {
    list<string> nom;
    list<string> ville;
    list<string>::iterator de;
    fstream entree;
    string ligne;
    // Vérifier les arguments, s'ils sont présents,
    // ouvrir le fichier sinon quitter le programme
    if (argc == 2)
        entree.open(argv[1]);
    else {
        cerr << "Erreur" << endl;
        exit(1);
    }
    // Lecture du fichier. Lire chaque ligne. La diviser
    // en deux. Et stocker les éléments dans deux listes
    getline(entree, ligne);
    while (entree) {
    int c = ligne.find(",");
        ville.push_back(ligne.substr(0, c));
        nom.push_back(ligne.substr(c+1, ligne.length()));
        getline(entree, ligne);
    }
    entree.close();
    // Affichage à des fins de test seulement
    cout << "Nbre = " << ville.size() << endl;
    de = ville.begin();
    for (int i = 0; de != ville.end(); de++)
        cout << i++ << ": " << *de << endl;
    de = nom.begin();
    for (int i = 0; de != nom.end(); de++)
        cout << i++ << ": " << *de << endl;
    return 0;
}
```

Entrée:

data.tx[ville, nom]

```
montreal, canadien
ottawa, senateurs
toronto, maple leafs
calgary, flames
vancouver, canuks
```

Sortie

exo4.exe data.txt

```
Nbre = 5
0: montreal
1: ottawa
2: toronto
3: calgary
4: vancouver
0:  canadien
1:  senateurs
2:  maple leafs
3:  flames
4:  canuks
```

Exercice 5 (30 points) Une des difficultés rencontrées lors de la création d'une entreprise est de trouver le bon groupe de financiers pour qu'il participe au financement de votre entreprise. Il y a un paquet de personnes avec une somme colossale d'argent à « jeter » par les fenêtres ... comment les persuader?
Vous allez résoudre ce problème en écrivant quelques classes en C++ pour simuler les différents types de financiers et leurs idiosyncrasies (tempérament personnel).
On catalogue 4 types de financiers :

- Type CR : C'est un financier au Capital Risque (CR). Il va investir dans tout sauf :
a) Il ne va pas investir dans une affaire où la compagnie a une valeur inférieure à 5,000,000$.
b) Il ne va pas investir dans une affaire où l'investissement est inférieur à 1,000,000$.
c) Il ne va pas investir dans une affaire qui a déjà 3 autres financiers.

- Type GCR : C'est un financier qui appartient à un Groupe au Capital Risque (GCR). Ce groupe, il a un nom établi. De ce fait, il s'est donc fixé des standards plus rigoureux :
a) Il ne va pas investir dans une affaire où la compagnie a une valeur inférieure à 8,000,000$.
b) Il ne va pas investir dans une affaire où un financier du type « CR » a investi dedans.
c) Sinon, il a les mêmes règles que celles d'un financier du type CR.

- Type Ange : C'est une personne qui a de l'argent et qui aimerait le distribuer à gauche et à droite.
a) Elle ne va pas financer une affaire où son investissement est supérieur à 500,000$.
b) Sinon, elle va participer à n'importe quel type d'investissements.

- Type OncleRiche : Votre oncle vous a promis depuis votre jeune âge qu'il allait participer au financement de votre entreprise.
a) Il va financer jusqu'à un plafond de 100,000$.
b) Il ne va financer qu'avec d'autres associés impliqués.

5.1 Dessiner, tout en expliquant votre démarche, un simple diagramme hiérarchique (boites/flèches) montrant la relation entre les classes « `Financiers` » (classe de base), « `CR` », « `GCR` », « `Ange` », « `OncleRiche` ». Ne pas ajouter d'autres classes. Se contenter de n'utiliser que ces 5 classes.

5.2 Pour chaque classe, autre que la classe « Financiers », implémenter la méthode « FaireAffaire ».

```
/* FaireAffaire :
       Détermine si un financier va investir en tenant compte des divers
       scénarios.

    Valeur de Retour : un booléen « True » (vrai) si le financier a décidé
d'investir.
    Arguments :
     - « valeurTotale » : La valeur de la compagnie en $
     - « maPart » : La contribution de ce financier en $
     - « Autres » : Les autres financiers potentiels
*/

virtual bool FaireAffaire(int valeurTotale, int maPart, vector<Financiers*> Autres);
```

5.3 Compléter l'implémentation de la fonction « `AffaireVaMarcher` ».

```
/*
      Valeur de retour : un booléen « True » (Vrai) si le marché proposé est
      accepté par tous les financiers.

      Arguments :
            Un entier représentant la valeur retenue de l'investissement,
            Un vecteur contenant les financiers potentiels,
            Un vecteur d'entiers contenant la contribution de chaque
            financier.
*/
```

Nous suggérons l'utilisation de la méthode « `insert()` » de la classe « `vector` », définie comme suit :

```
void vector<T>::insert(Iterator PointInsertion, Iterator Debut, Iterator Fin);
```