Trimestre Hiver, 2011

Mohamed Lokbani

IFT1169 – Examen Final –

Inscrivez tout de suite : votre nom et le code permanent.

Nom : _____| Prénom(s) : _____|

Signature : _____| Code perm : _____|

Date : mardi 12 avril 2011

Durée : 3 heures (de 18h30 à 21h30)

Local : Z-200 ; Pavillon Claire McNicoll

Directives :

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

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

1._____/10 (1.1, 1.2, 1.3, 1.4, 1.5)

2._____/20 (2.1, 2.2, 2.3)

3._____/20 (3.1 3.2 3.3)

4._____/20 (4.1 4.2 4.3)

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)

Soit le fichier « makefile » suivant :

```
 1  # Fichier makefile exercice -1-
 2
 3  CXX = g++
 4  CXXFLAGS = -Os -Wall
 5  OPT = -pedantic
 6
 7  OBJS = un.o deux.o prog.o
 8
 9  default: prog
10
11  prog: $(OBJS)
12          $(CXX) -o prog.exe $(OBJS)
13
14  un.o: un.cpp
15          $(CXX) $(CXXFLAGS) $(OPT) -c un.cpp
16
17  deux.o: deux.cpp
18          $(CXX) $(CXXFLAGS) $(OPT) -c deux.cpp
19
20  prog.o: prog.cpp
21          $(CXX) $(CXXFLAGS) $(OPT) -c prog.cpp
```

1.1 Expliquez en quelques mots l'utilité du fichier « makefile ».

1.2 Donnez toutes les commandes générées suite au lancement de la commande « make » dans un terminal.

1.3 Supposons maintenant, après avoir lancé la commande « make » à la question « 1.1 », que vous ayez décidé d'introduire des modifications dans le fichier « prog.cpp ». Donnez toutes les commandes générées suite au lancement de la commande « make » dans un terminal.

1.4 Vous avez décidé maintenant d'apporter des modifications dans le fichier « un.h » et vous avez lancé une nouvelle fois la commande « make ». Que se passe-t-il? Quel devrait être le comportement normal? Apportez les modifications dans le fichier « makefile » si nécessaire.

1.5 Ajoutez à ce « makefile » une règle qui fait le ménage. En plus d'éliminer les fichiers qui ne vous intéressent pas pour une raison ou une autre, elle se charge aussi d'éliminer l'exécutable.

Exercice 2 (20 points)

2.1 Décrivez la différence entre lire un fichier avec l'opérateur « >> » et avec la méthode « getline » membre de la classe « fstream ». Étayez vos explications à l'aide d'un simple exemple (juste les instructions nécessaires, pas besoin d'écrire tout un programme).

2.2 Quel est le nom de cette constante qui se met à 1, quand une opération de lecture atteint la fin d'un fichier. À l'aide de quoi pourriez-vous vérifier l'état d'une telle constante?

2.3 Tout en détaillant votre réponse, que va afficher en sortie le programme suivant qui compile et s'exécute correctement?

```
1  #include <iostream>
2  #include<fstream>
3
4  using namespace std;
5
6  int main() {
7          int i = 0;
8          int x;
9          ifstream f;
10         f.open("entree.dat");
11         do {
12                 f >> x;
13                 if (!f.eof()) {
14                         i++;
15                         if (i >= 4) cout << x << ' ';
16                 }
17         } while (!f.eof());
18         f.close();
19         cout << "\n";
20         return 0;
21 }
```

Le fichier « entree.dat » contient ce qui suit :

```
7 8
-18 64 -37
12 9
```

Exercice 3 (20 points)

3.1 Expliquez à quoi sert l'instruction suivante dans un programme « C++ » :

template <class T> Tableau <T>::Tableau(int s)

3.2 Dans le langage « C++ », la notion de polymorphisme est mise en œuvre à l'aide de fonctions virtuelles. La fonction à utiliser est choisie au moment de l'exécution du programme.

Expliquez pourquoi les fonctions (et classes) templates sont considérées comme une forme (primaire) de polymorphisme?

3.3 A) Écrivez la classe template « Paire » qui peut instancier deux valeurs « X » et « Y » du type, respectivement, « T » et « U ». Cette classe doit contenir juste le constructeur et deux méthodes simples permettant d'accéder à ces valeurs (des « get »).

B) Écrivez la classe template « Triple » qui peut instancier trois valeurs : « X », « Y » de « Paire » et « Z ». Ces trois valeurs sont respectivement du type « T », « U » et « V ». Cette classe doit dériver de la classe « Paire ». Elle va contenir comme la classe « Paire », juste le constructeur et les méthodes nécessaires permettant d'accéder à ses valeurs (des « get »).

C) Finalement écrivez la fonction « main » qui crée et utilise une instance de « Paire » et une instance de « Triple ». L'instance de « Paire » doit contenir un entier et un « string ». L'instance de « Triple » doit contenir un caractère, un réel (double) et un booléen. Initialiser ces instances avec des valeurs fictives. Puis faites appel dans aux différents « get » et affichez les valeurs des deux instances en sortie. Ajoutez les commentaires appropriés à chaque étape, permettant d'expliquer les opérations réalisées dans la fonction « main ».

Exercice 4 (20 points)

4.1 Pour le fragment de code ci-dessous :

```
1  char **xyz;
2  char a[20] = "boomerang";
3  char *b = "musique";
4  char *d = new char[20];
```

Dites dans quelle zone mémoire (tas ou pile) sont situées les variables suivantes :

xyz:

a[2] :

d :

d[4] :

4.2 Expliquez pourquoi une fuite de mémoire est susceptible d'avoir lieu dans le fragment de code suivant :

```
1  void g(){
2      throw "Exception";
3  }
4
5  void f() {
6      int* i = new int(0);
7      g();
8      delete i;
9  }
10
11 int main() {
12     f();
13    return 0;
14 }
```

Ajoutez les correctifs nécessaires dans la fonction « f » pour que cette fuite mémoire soit colmatée. Attention : l'appel à la fonction « g » dans la fonction « f » doit avoir lieu quand même. Vous n'allez pas me supprimer cet appel!

4.3 Tout en expliquant votre démarche, dites que va afficher en sortie le programme suivant qui compile et s'exécute correctement.

```
1  #include <iostream>
2
3  using namespace std;
4  void UneFonction( void );
5
6  class Test {
7  public:
8      Test(){};
9      ~Test(){};
10     const char *DonnerUneRaison() const { return "Exception dans \
11                                           la classe Test."; }
12 };
13
14 class Demo{
15 public:
16     Demo();
17     ~Demo();
18 };
19 Demo::Demo(){
20     cout << "Construction de démo." << endl;
21 }
22 Demo::~Demo(){
23     cout << "Destruction de démo." << endl;
24 }
25 void UneFonction(){
26     Demo D;
27     cout<< "Dans UneFonction(). Exception Test est levèe:" << endl;
28     throw Test();
29 }
30 int main(){
31     cout << "Dans main." << endl;
32     try{
33         cout << "Dans le bloc try, appel à UneFonction()." << endl;
34         UneFonction();
35     }
36     catch( Test E ){
37         cout << "Dans catch." << endl;
38         cout << "Capture d'une exception du type Test: ";
39         cout << E.DonnerUneRaison() << endl;
40     }
41     catch( char *str ){
42         cout << "Capture une autre exception: " << str << endl;
43     }
44     cout << "retour au main. Suite des opérations." << endl;
45     return 0;
46 }
```

Exercice 5 (30 points)

5.1 Stocker les éléments suivants {3871, 2533, 8243, 6489, 6354, 4589, 2979} (dans cet ordre) dans une table de hachage préalablement vide, de taille « 10 ». Vous allez utiliser pour cela la fonction de hachage « `h(x) = x%10` », et la technique de la première case libre à droite (après).

Valeur	Indiquez les positionnements successifs dans le tableau
3871	
2533	
8243	
6489	9
6354	
4589	9, 0
2979	

Pour « 4589 » : position « 9 », mais comme elle est occupée déjà par « 6489 », il va s'insérer dans la position « 0 ».

0	1	2	3	4	5	6	7	8	9
4589									6489

5.2 Nous allons placer les éléments en une technique du double hachage, en utilisant l'algorithme suivant :

Pour l'élément « i » :

1) Calculer son index en utilisant la fonction de hachage de « 5.1 » i.e. : h(x) = x % 10
2) Si l'index est libre, placez l'élément dans le tableau, opération de placement pour « i » prend fin.
3) Sinon calculer la nouvelle valeur comme suit :
   - h2(x) = 7 – (x % 7)
   - nouveau index = h(x) + h2(x)
   - Si la valeur obtenue est inférieure à 10, placer l'élément dans le tableau. Si la place est occupée, allez à l'étape « 3 » avec la nouvelle valeur.
   - Si la valeur est supérieure à 10, allez à l'étape 1 avec cette nouvelle valeur.

Finalement, après avoir calculé les indexes des différentes valeurs, insérez les dans le tableau. Si aucune place n'est libre pour une valeur donnée, juste mentionnez « pas de place ».

Valeur	Indiquez les positionnements successifs dans le tableau
3871	
2533	
8243	
6489	
6354	
4589	
2979	

0	1	2	3	4	5	6	7	8	9

5.3 La fonction « LeVainqueurEst » a la signature suivante :

```
string LeVainqueurEst( const vector<string> &bulletins);
```

Cette fonction prend comme argument un vecteur contenant les noms apposés sur les bulletins de votes déposés dans l'urne et elle retourne le nom du vainqueur si ce dernier a obtenu strictement plus de la moitié des voix. Dans le cas contraire elle retourne une chaine vide.

Si « X », « Y » et « Z » sont les noms des candidats qui se sont présentés à cette élection, voici le comportement de la fonction « LeVainqueurEst » dans 3 cas de figures :

```
1 : urne {"X","Y","Z","Y","X","Y","Y","Z","Y"} – Le vainqueur est "Y"
2 : urne {"X","X","X","Z","Z","Y","Y","Z","Z","Z","Y","Z","Z"} – Le vainqueur est "Z"
3 : urne {"X","Y","Z","Y","X","Y","Z","Y"} – pas de vainqueur ""
```

a) Écrivez le code de la fonction « LeVainqueurEst » sans utiliser de « map » :

b) Écrivez cette fois-ci le code de la fonction « LeVainqueurEst » en utilisant obligatoirement une « map », dont la clé est le nom du candidat et dont la valeur est le nombre de voix obtenues.

Bon été!