IFT1166 - Session Été 2000, Final

Mohamed Lokbani

Nom:_____| Prénom(s):_____|

Signature:_____| Code perm:_____|

Date: 27 juin 2000

Durée: 3 heures (de 18h30 à 21h30) Local: P-310

Directives:

- Toute documentation permise.
- Calculatrice et ordinateurs personnels prohibés.

- Répondre directement sur le questionnaire.

Bon courage!

1._____/15 Q1

2._____/15 Q2: Er1, Er2, Er3, Er4, Er5

3._____/20 Q3: li1, li2, li3, li4, li5

4._____/30 Q4: 4.1, 4.2, 4.3, 4.4

5._____/20 Q5

Total:_____/100

Bonus._____/10 B

Question 1 (15 points)

Soit le programme suivant, qui compile et s'exécute correctement:

```
// début
#include <iostream.h>
void transfert(ifstream *entree,ofstream *sortie){
  char ch;
  bool flag = false;
  while (entree->get(ch)) {
    if (ch == ' ' && !flag) {
      flag = true;
      *sortie << ch;
    } else if (ch != ' ') {
      flag = false;
      *sortie << ch;
    }
  }
}
int main() {
  ifstream ficentree("entree.txt");
  ofstream ficsortie("sortie.txt");

  transfert(&ficentree,&ficsortie);

  return 0;
}
// fin
```

Tout en développant votre réponse sur la feuille d'examen, indiquez ce que va contenir le fichier `sortie.txt` après exécution du programme précédent, si le fichier `entree.txt` contient:

```
Deux□équipes de football, composées chacune de  onze vaches portant dos
sard numéroté et trottinant sur un faux pré de  couleur orange situé le
   long de l'autoroute entre    Amsterdam et Utrecht,  ont  provoqué un
  embouteillage de plus de  douze kilomètres  mardi.▲
```

Note: □ représente un espace blanc. ▲ représente une fin de ligne. Ils doivent être représentés dans votre réponse, si nécessaire.

Question 2 (15 points)

Soit le programme suivant:

```
// début
 1) #include <iostream.h>

 2) class Droite {
 3)   Coordonnee gauche; // extrémité gauche
 4)   Coordonnee droite; // extrémité droite
 5) public:
 6)   Droite(int gx,int gy,int dy):gauche(gx,gy),droite(dx,dy){}
 7)   Droite(const Droite& d)gauche(d.gauche),droite(d.droite) {}
 8)    bool operator==(const Droite& d2){
 9)      if ((gauche==d2.gauche) && (droite==d2.droite))
10)          return true;
11)     return false;
12)   }
13) };
14) class Coordonnee {
15)   int x;
16)   int y;
17) public:
18)   Coordonnee(int a, int b): x(a);y(b){}
19)   Coordonnee(const Coordonnee& c) {
20)     x = c->x; y = c->y;
21)   }
22)   bool operator==(const Coordonnee& c) {
23)     if ((x==c.x) && (y==c.y)) return true;
24)     return false;
25)   }
26) };
27) int main() {
28)   Droite d1(2,3,4,5);
29)   Droite d2(7,8,9,10);
30)   Droite d3(d1);
31)   cout << (d1==d3) << (d2==d3) << (d1==d2) << endl;
32)   return 0;
33) }
// fin
```

Le programme contient 5 erreurs de syntaxe. Indiquez les et indiquez la correction qu'il faudra apporter. À signaler les numéro de lignes ne font pas partie du programme, elles sont la à titre indicatif. Des erreurs sur une même ligne compte pour une seule erreur.

Erreur 1:

Erreur 2:

Erreur 3:

Erreur 4:

Erreur 5:

Question 3 (20 points)

Soit le programme suivant, qui compile et s'exécute correctement:

```
// début
#include <iostream.h>

template <class T> T fonction(int a,T* b) {
  cout << "template 1\n";
  return b[0];
}
template <class T> T fonction(T a,T b) {
  cout << "template 2\n";
  return a;
}
double fonction(int a,char* b) {
  cout << "fonction 1\n";
  return a;
}
double fonction(double a,double b) {
  double x=4.5;
  cout << "fonction 2\n";
  return x;
}
int main() {
  char c;
  char chaine[20];
  unsigned int tab[20];
  int a;
  double i,j;

  fonction(i,j);                      // ligne -0-
  fonction(a,&c);                     // ligne -1-

  a = fonction(tab[0],tab[1]);        // ligne -2-

  fonction(tab[0],c);                 // ligne -3-

  fonction(c,tab);                    // ligne -4-

  fonction(c,chaine);                 // ligne -5-

  return 0;
}
// fin
```

Le compilateur va associer à chaque fonction appelée entre les lignes 0 et 5, une fonction classique ou une fonction instanciée à partir d'une fonction générique. À titre d'exemple, pour la ligne 0, la fonction prise en considération sera celle qui affichera en sortie "`fonction 2`":

ligne -0- `fonction(i,j);`
Résultat affiché: `fonction 2`
Pourquoi? blablabla …

En suivant le même principe, complétez ce qui suit:

ligne -1- `fonction(a,&c);`
Résultat affiché:
Pourquoi?

ligne -2- `a = fonction(tab[0],tab[1]);`
Résultat affiché:
Pourquoi?

ligne -3- `fonction(tab[0],c);`
Résultat affiché:
Pourquoi?

ligne -4- `fonction(c,tab);`
Résultat affiché:
Pourquoi?

ligne -5- `fonction(c,chaine);`
Résultat affiché:
Pourquoi?

Question 4 (30 points)

Dans cette exercice vous allez programmer une représentation d'un carnet d'adresses postales. Une adresse postale, peut avoir différents formats ; par exemple, pour un envoi international du Canada vers les USA, nous devons préciser le pays receveur de la lettre (ici USA) alors que pour un envoi à l'intérieur du Canada cela n'est pas nécessaire. Le format d'écriture de l'adresse est, quant à lui, dépendant du pays expéditeur. Une représentation simplifiée de ce format est comme suit:

à partir des USA	**à partir du Canada**
Nom de la personne Numéro et le nom de la rue Ville, État, et Code postal {Pays}	Nom de la personne Numéro et le nom de la rue Ville {Pays} Code postal

tableau -1-

Les { } indiquant un affichage optionnel: i.e. le pays ne sera pas nécessairement affiché pour un envoi interne.

Techniquement, ce carnet sera représenté par 4 classes, comme le montre le schéma d'héritage suivant:

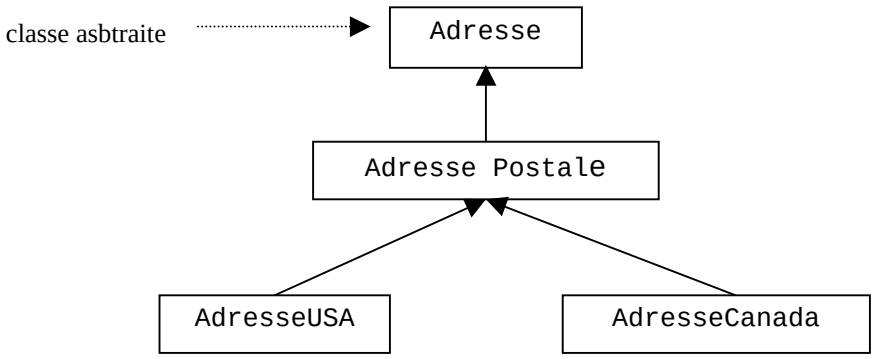

figure -1-

Q4.1 Définir la classe abstraite `Adresse` qui contient parmi ses membres les 3 fonctions virtuelles pures et publiques suivantes:

- `type`: sans argument, et qui retourne une chaîne de caractères spécifiant le type d'adresse auquel nous avons affaire (adresse postale, adresse courriel etc.).

- `affiche`: accepte un argument du type `int` et ne retourne rien. L'argument permet de spécifier si on a affaire à un envoi international, si c'est le cas, le pays destinataire sera affiché. Par défaut cet argument prendra la valeur 0, pour signifier que c'est un envoi intérieur. Cette fonction sert à afficher l'adresse postale comme indiqué dans le tableau -1-.

- `saisir`: sans argument et qui ne retourne aucune valeur. Cette fonction permet de saisir des données du clavier.

Q4.2 Définir la classe AdressePostale qui dérive publiquement de la classe Adresse, et qui doit respecter les contraintes suivantes:

- le principe d'encapsulation des données doit être respecté.
- le constructeur doit être accessible que par les classes dérivées.
- `AdressePostale` doit contenir les membres données suivants:

`char* nom;` // nom de la personne	`char* rue;` // nom de la rue
`char* numero;` // numéro de la maison	`char* ville;` // nom de la ville
`char* codepostale;` // le code postale	`char* pays;` // nom du pays

- pour chaque membre donnée, vous devez définir deux méthodes:

  `saisir`: accepte un argument du type chaîne de caractères et ne retourne rien. Cette fonction permet de passer l'information au membre donnée. À titre d'exemple pour le membre donnée `nom`, cette fonction portera le nom de: `saisirnom` (pour `numero`: `saisirnumero` etc.). Au total vous devez écrire 6 fonctions "saisir". Lors de la correction, il sera tenu compte de l'optimisation de votre code.

  `affiche`: sans argument ne retourne rien. Cette fonction permet d'afficher l'information sur le membre donnée concerné. À titre d'exemple pour le membre donnée `nom`, cette fonction portera le nom de: `affichenom` (pour `numero`: `affichenumero` etc.). Au total vous devez écrire 6 fonctions "affiche".

- la définition de la fonction `type`, décrite en Q4.1.

De la classe `AdressePostale` vont dériver deux classes: `AdresseCanada` et `AdresseUSA` (voir figure -1-, à la page 8).

Q4.3. Écrire la définition de la classe `AdresseUSA`. Parmi ses fonctions membres, les fonctions: `affiche` et `saisir` décrites en Q4.1 et en tableau -1- et que vous devez définir à ce niveau.

Q4.4. Écrire la définition de la classe `AdresseCanada`. Parmi ses fonctions membres, les fonctions: `affiche` et `saisir` décrites en Q4.1 et en tableau -1- et que vous devez définir à ce niveau.

Question 5 (20 points)

En utilisant les STL, écrire un programme qui manipule un vecteur d'entiers. Votre programme doit:

* déclarer le tableau suivant:

```
int tab[] = {1,2,3,4,5,6,7,8,9,10};
```

* définir ce qui suit:

- 2 fonctions globales:
  - **`carre:`** elle permet de calculer le carré d'un entier passé par référence, et retourne le résultat.
  - **`affiche`** elle permet d'afficher sur la sortie standard un entier, passé en argument.
- une classe:
  - **`additionne`**: suite à son appel à travers l'opérateur d'appel de fonction `()`, elle modifie la valeur d'un entier en lui additionnant le carré d'une valeur entière donnée (par défaut égale à 10), passée en paramètre. Le carré de cette valeur est calculé en utilisant la fonction `carre` précédemment définie.
- la fonction principale:
  - **`main`**: elle utilise la classe et les fonctions globales pour manipuler le tableau `tab`, en s'aidant dans cette tache de la batterie d'algorithmes définis dans les STL.

* afficher en sortie, après l'avoir exécuté, le résultat suivant:

```
Carré des 10 premiers entiers: 1 4 9 16 25 36 49 64 81 100
Additionne le carré de 100 à chaque élément du tableau:
        10001 10004 10009 10016 10025 10036 10049 10064 10081 10100
Additionne le carré de la valeur par défaut à chaque élément du tableau:
        10101 10104 10109 10116 10125 10136 10149 10164 10181 10200
```

Dans cet exercice, il vous est interdit d'utiliser les structures de contrôle suivantes: `if, while, for, do/while, swtich/case, break, continue`.

Bonus (10 points)

Par de simples opérations mathématiques sur des entiers non signés (unsigned int), vous allez définir les outils nécessaires pour lever les exception: `overflow` et `underflow`.

- Ces exceptions vont agir sur les 3 fonctions mathématiques suivantes:
  - **`add`**: additionne deux entiers non signés passés en argument (i.e. `a+b`), et retourne le résultat de l'opération. Cette fonction peut lever que l'exception du type `overflow` si le résultat de l'opération d'addition est inférieur au premier argument (i.e. `a`) ou bien au second argument (i.e. `b`).
  - **`sous`**: soustrait deux entiers non signés passés en argument (i.e. `a-b`), et retourne le résultat de l'opération. Cette fonction peut lever que l'exception du type underflow si le résultat de l'opération d'addition est supérieur au premier argument (i.e. `a`).
  - **`calc`**: calcul une simple opération mathématique (i.e. `a+(b-c)`) en utilisant 3 entiers non signés passés en argument, et retourne le résultat. Cette opération mathématique est calculée en utilisant pour cela les fonctions préalablement citées (`add` et `sous`). Cette fonction lèvera les deux types d'exception précédemment définies: `overflow` et `underflow`.

- Chaque exception sera représentée par une structure, ayant les deux membres suivants:
  - **`fonctmath`**: nom de la fonction mathématique qui a levé l'exception (ex. `add`, `sous`, ou `calc`).
  - **`msgexp`**: message signalant que l'exception (son nom) a été levée par la fonction mathématique (son nom), ex.: "`overflow dans add`", "`underflow dans sous`".

Vote tache consiste à définir l'ensemble des fonctions et des structures précédemment décrites et complétez la fonction main suivante:

```
int main() {
   unsigned int a=23,b=34,c=45;
   try {
          unsigned int resultat = calc(a,b,c);
          cout << "le résultat est: " << resultat << endl;
          return 0;
      }
      // à compléter par vous …

}
```