IFT1166 - Session Automne 2000, Final

Mohamed Lokbani

Date: 20 décembre 2000 Durée: 3 heures (de 18h30 à 21h30) Local: Z-210

## Directives:

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

L'examen compte: 6 pages (incluant celle-ci) et 5 exercices.

**Bon courage!**

## Question 1 (15 points)

Que va afficher le programme suivant en sortie suite à cet appel (supposé que la version executable du programme se nomme prg):

```
./prg test chaine
```

```
#include <iostream.h>

void  une_certaine_fonction(char* arg1,char* arg2){
      cout << "arg1: "<< arg1 << " ; arg2: " << arg2 << endl;
}
int main(int argc, char* argv[]) {

      if (argc < 3) {
            cerr << "Manque un ou plusieurs arguments" << endl;
            cerr << "Commande: ./prg arg1 arg2" << endl;
            exit(1);
      }
      une_certaine_fonction(argv[1],argv[2]);

      return 0;
}
```

## Question 2 (15 points)

En tenant compte de la fonction main suivante, écrire la fonction générique **`conversion`** qui permet de convertir un caractère (**`char`**) en un entier (**`int`**), convertir un réel (**`double`**) en un entier (**`int`**) et convertir un entier (**`int`**) en un réel (**`double`**).

```
int main() {

      char c = 'A';
      int e;

      conversion(c,e);
      cout << c << " ; " << e << endl; // devrait afficher: A ; 65

      double d=4.15;

      conversion(d,e);
      cout << d << " ; " << e << endl; // devrait afficher: 4.15 ; 4

      conversion(e,d);
      cout << d << " ; " << e << endl; // devrait afficher: 4 ; 4

      return 0;

}
```

## Question 3 (20 points)

Soit la classe suivante:

```
class Chaine{
     char* nom;
     int taille;
public:
     // supposez que les 4 fonctionnalités suivantes sont disponibles
     Chaine(char*);
     Chaine(const Chaine&);
     Chaine& operator=(const Chaine&);
     ostream& operator<<(ostream& out,const Chaine&);
};
```

Complétez la classe `Chaine` par les opérateurs suivants:

**operator+=(char)**: pour ajouter un caractère à la fin du membre de la classe `Chaine`: `nom`. Par exemple si l'objet `x`, instance de la classe `Chaine`, contient au départ la chaîne **test** alors:

```
      x+=('e');
      cout << x;
```

affichera **teste** en sortie.

**`operator-=(int)`**: pour retirer un certain nombre de caractères de la fin du membre de la classe `Chaine`: `nom`. Par exemple si l'objet `x`, instance de la classe `Chaine`, contient au départ la chaîne **teste** alors:

```
      x-=(2);
      cout << x;
```

affichera **tes** en sortie.

## Question 4 (15 points)

En utilisant uniquement les **4 fonctions** `append`, `assign`, `sort` et `reverse` de la `STL` et de la classe `string` ; écrivez un programme qui en acceptant les deux chaînes de caractères suivantes (type `string`):

```
"Les tableaux de Monet seront exposés à partir de la semaine prochaine."
```

```
"Les meilleurs livres sont ceux qui racontent ce que l'on sait déjà."
```

D'abord crée la phrase suivante (du type `string`):

```
"Les meilleurs livres seront exposés à partir de la semaine prochaine."
```

puis à partir de cette phrase génère les deux phrases suivantes (du type `string`):

```
".eniahcorp eniames al ed ritrap à sésopxe tnores servil sruelliem seL"
```

```
"àé          .Laaaacdeeeeeeeeeehiiiiillllmmnnnoooppprrrrrrsssssssttuvx"
```

### Note:

Vous ne devez pas utiliser les structures de contrôle (if, for, while, do/while etc.) ni les itérateurs.

## Question 5 (35 points)

On définit une hiérarchie de formes géométriques comportant le cercle, le triangle, et le carré. Chaque forme est représentée dans le plan par ses coordonnées (x,y). Pour chaque forme on veut connaître son périmètre et sa surface.

| forme | périmètre | surface |
|---|---|---|
| `cercle` | `2 * 3.14 * rayon` | `3.14 * rayon * rayon` |
| `rectangle` | `2 * (largueur + longueur)` | `largueur * longueur` |
| `carré` | `4 * coté` | `coté * coté` |

**Q5.A** Écrire les classes nécessaires à l'implémentation de la hiérarchie suivante: La classe **`Cercle`** dérive publiquement de la classe **`Forme`**, la classe **`Rectangle`** dérive publiquement de la classe **`Forme`**, et finalement la classe **`Carre`** dérive publiquement de la classe **`Rectangle`** ; en vous basant pour cela de la fonction main ci-jointe ainsi que du résultat obtenu après exécution du programme.

**Exemple**

Soit la déclaration suivant: `Cercle cercle(10,10,4);` `cercle` est un objet dont les cordonnées dans le plan sont (10,10), son rayon est 4, son périmètre est égal à 25.12 (2*3.14*4) et sa surface est égale à 50.24 (3.14*4*4).

**Fonction principale main**

```
int main() {

   Cercle cercle(10,10,4);
   cout << endl << cercle << " surface=" << cercle.surface() << endl;
   cout << cercle << " perimetre=" << cercle.perimetre() << endl <<
   endl;

   Rectangle rectangle(30,30,2,5);
   cout << rectangle << " surface=" << rectangle.surface() << endl;
   cout << rectangle << " perimetre=" << rectangle.perimetre() << endl
   << endl;

   Carre carre(100,100,2);
   cout << carre << " surface=" << carre.surface() << endl;
   cout << carre << " perimetre=" << carre.perimetre() << endl << endl;

   return 0;
}
```

**Sortie après exécution du programme**

```
Cercle (10,10) r=4 surface=50.24
Cercle (10,10) r=4 perimetre=25.12

Rectangle (30,30) Longueur=2 Largeur= 5 surface=10
Rectangle (30,30) Longueur=2 Largeur= 5 perimetre=14

Carre (100,100) Cote=2 surface=4
Carre (100,100) Cote=2 perimetre=8
```

**Q5.B** Modifiez le contenu des classes précédemment définies pour tenir compte de la fonction main suivante, qu'il faudra compléter, et de la sortie obtenue (voir page suivante) suite à l'exécution du programme.

Chaque méthode nécessitant des modifications doit être réécrite complètement.

```
int main() {

   Cercle cercle(10,10,4);
   cout << endl << cercle << " surface=" << cercle.surface() << endl;

   Rectangle rectangle(30,30,2,5);
   cout << endl << rectangle << " surface=" <<\
                                          rectangle.surface() << endl;

1  // à faire: allocation dynamique d'un carré(100,100,2):
   ……

   cout << endl << *carre << " surface=" << carre->surface() << endl;

2  …… tab[3]; // déclaration du tableau tab à compléter
              // devant contenir ce qui suit:

   tab[0] = &cercle;
   tab[1] = &rectangle;
   tab[2] = carre;


   float surf=0.0;

   for (int i=0; i<3; i++)  {

      // à faire: surf calcule la somme totale des surfaces

3     surf += ……;

   }
   cout << endl << "surface totale : " << surf << endl << endl;
   cout << "périmètre d'une forme tirée au hasard" << endl;

   srand((unsigned int) time(NULL));


   // à faire: définition de ptr: un pointeur qui pointe un des
   // éléments du tableau tab. L'élément est choisi au hasard.
4  …… ptr = tab[rand()%3];

   // à faire: compléter les deux champs, basez-vous pour cela sur
   // l'affichage en sortie (voir plus bas).

5  cout << ……<< " : périmètre="\
6                    << …… << endl << endl;


   cout << "destruction de carré alloué dynamiquement" << endl;

   ptr = carre;
   delete ptr;
   cout << endl;

   return 0;

}
```

**Sortie Q5.B après exécution du programme**

```
- Forme::Forme -- Cercle::Cercle -
Cercle (10,10) r=4 surface=50.24
- Forme::Forme -- Rectangle::Rectangle -
Rectangle (30,30) Longueur=2 Largeur= 5 surface=10
- Forme::Forme -- Rectangle::Rectangle -- Carre::Carre -
Carre (100,100) Cote=2 surface=4

surface totale : 64.24

périmètre d'une forme tirée au hasard
Rectangle (30,30) Longueur=2 Largeur= 5 : périmètre=14

destruction de carré alloué dynamiquement
- Carre::~Carre -- Rectangle::~Rectangle -- Forme::~Forme -

- Rectangle::~Rectangle -- Forme::~Forme -
- Cercle::~Cercle -- Forme::~Forme -
```

**Q5.C** Dans votre réponse à la question précédente (Q5.B) était-il nécessaire d'utiliser le polymorphisme? Expliquez.

---------- Fin de l'examen ----------