TP-7 POO en C++ (7ème et 8ème Séance)

Ce TP aborde les points suivants :

- **Surcharge d'opérateur de la classe String !**
- **Héritage simple, Héritage multiple, Polymorphisme**
- **Fonction virtuelle pure et classe abstraite**
- **Manipulations des fonctions des flux (open, read, write, close, seek, tell), avec Héritage multiple et notion classe générique (Template)**
- **Surcharge d'opérateur de la classe String !**

Chapitre 5, 6, 7 du cours

Exercice 1 : Classe String avec Surcharge des opérateurs

On désire modifier la classe String de l'exercice précédent pour y ajouter les opérateurs :

= [] + += << >> == !=

1. Ecrire la nouvelle déclaration de cette classe dans un fichier ***string2.h***.
2. Après avoir fait valider cette déclaration, vous devez entreprendre la définition des méthodes dans un fichier ***string2.cpp***.
3. Ensuite utiliser le programme essaiString2.cpp suivant pour valider la classe

```
#include <iostream.h>
#include "string2.h"
void main() {
    String ch1("essai"), ch2 = ch1, ch3('=', 80);
    const String ch4("chaîne de caractères constante");

    ch1[1] = 'E';    // le premier caractère de la chaîne
    cout << ch4[1] << endl;
    ch1 = "<<<< " + ch2 + " >>>>";
    cout << ch1 << endl;
    cin >> ch2;
    ch2 += " de la classe Strin";
    ch2 += 'g';
    if ( ch2 != "" ) cout << ch2 << endl;
    cout << ch2.minuscule() << endl;
}
```

Exercice 2 : Classe Forme : héritage et polymorphisme

1. Préparation :

On définit une hiérarchie de formes géométriques : le cercle, le triangle, le rectangle et le carré. Pour chaque forme, on veut connaître son périmètre et sa surface. Il possible de déplacer une forme dans le plan, on définira pour cela une classe Coordonnees.

2. Ecrire les classes : nécessaires à l'implémentation de cette hiérarchie en vous basant sur l'extrait du « main » ainsi que le résultat de l'exécution fournis ci-dessous.

Extrait du main :

```
void main() {
     Cercle cercle(10,10,4);
      cout <<endl<<cercle<<" surface="<<cercle.surface()<<endl;
      Triangle triangle(20,20,3);
      cout<<endl<<triangle<<" surface="<<triangle.surface()<<endl;
      Rectangle rectangle(30,30,2,5);
      cout << endl <surface() << endl << endl;
      cercle.deplace(50,50);
      cout << "déplacement " << endl;
      cout << cercle << endl << endl;

     *********;   //déclaration du tableau initialisé ci-dessous ???

      tab[0] = &cercle;
     tab[1] = &triangle;
     tab[2] = &rectangle;
     tab[3] = carre;
     float surf=0.0;
     for (int i=0; i<4; i++)  {
          surf += *****************;
     }
     cout << "surface totale : " << surf << endl << endl;

     cout << "périmètre d'une forme tirée au hasard" << endl;
     srand((unsigned int) time(NULL));
     ****** ptr = tab[rand()%4];  // définition de ptr
     cout << **** << " périmètre=" << ******** << endl << endl;
     cout << "destruction de carré alloué dynamiquement" << endl;
     ptr = carre; delete ptr; cout << endl;
}
```

Résultat d'exécution :

```
 - Forme::Forme -- Cercle::Cercle -
Cercle (10,10) r=4 surface=50.2655
- Forme::Forme -- Triangle::Triangle -
Triangle (20,20) c=3 surface=4.5
- Forme::Forme -- Rectangle::Rectangle -
Rectangle (30,30) L=2 l=5 surface=10
- Forme::Forme -- Rectangle::Rectangle -- Carre::Carre -
Carre (100,100) c=2 surface=4

déplacement
Cercle (60,60) r=4

surface totale : 68.7655

périmètre d'une forme tirée au hasard
Carre (100,100) c=2 périmètre=8

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

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

3 .Questions :

1. Où intervient le **polymorphisme** dans le **programme** ci-dessus ?
2. Expliquez l'ordre d'appel des constructeurs et destructeurs.

Exercice 3 : Classe FileOf

1. Objectifs :
   - héritage multiple
   - manipulations de base sur les flux (open, read, write, close, seek, tell)
   - classe générique
2. Préparation :

Ecrire les classes, de la hiérarchie ci dessous, permettant de manipuler des fichiers constitués d'entiers et donnant l'accès direct aux éléments du fichier.

3. Exemple d'utilisation :

```
int main() {
       W_File_of_Int f_out("essai.fic");

       if ( ! f_out ) {
         cerr << "erreur à la création de 'essai.fic'\n";
         return 1;
       }
```

```
for(i=0; i<=10; i++) // Ecriture de 11 entiers dans le fichier
       f_out << i;
cout << f_out.tellp() << "éléments sont écrits dans le fichier.\n";
         // affiche:  11 éléments sont écrits dans le fichier.
f_out.close();
R_File_of_Int f_in("essai.fic");
 int entier;
if ( ! f_in ) {
         cerr << "erreur à la création de essai.fic\n";
         return 1;
}
f_in.seekg(0, ios::end);  // se positionne à la fin du fichier
cout << "Il y a " << f_in.tellg() << " éléments dans le fichier.\n";
          // affiche:  Il y a 11 éléments dans le fichier.
f_in.seekg(0);      // se positionne au début du fichier
while ( 1 ) {        // affichage du contenu du fichier
    f_in >> entier;
    if ( f_in.eof() )
     break;
     cout << entier << " ";
}
f_in.clear(); // ne pas l'oublier ... sortie du while sur erreur eof
cout << endl;
f_in.close();

////////////////////////////////////////////////////////////////
RW_File_of_Int f_io("essai.fic"); //s'il existe, il n'est pas écrasé
if ( ! f_io ) {
    cerr << "erreur à la création de essai.fic\n";
     return 1;
}
f_io.seekp(0, ios::end);   // se positionne à la fin du fichier

cout<<"Il ya déjà"<<f_io.tellp() <<"éléments dans le fichier\n";
            // affiche:  Il y a déjà 11 éléments dans le fichier
for(i=11; i<=19; i++) f_io << i;
f_io.seekp(10);          // se positionne sur le 11 ième entier
while ( 1 ) {            // affichage du contenu du fichier
     f_io >> entier;
     if ( f_io.eof() ) break;
     cout << entier << " ";
}  // affiche:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
f_io.clear(); // ne pas l'oublier… sortie du while sur erreur eof
f_io.close();
return 0;
}
```

4. Remarques :
   - Une instruction du type f_out.write (...); ne doit pas être compilable.
   - La définition du constructeur de copie n'est pas demandée.
5. Généricité (Template) :
   - Rendre générique ces classes.