Examen Méthodologie Orientée Objets (IF2)

Durée 1h, documents autorisés,
Annales et appareils communiquant interdits

Exercice 1 : Trouver les erreurs (15 minutes)

1) Les 4 programmes suivants possèdent tous ***une*** erreur de conception (*i.e.* une erreur lors de l'exécution, résultat incohérent). Pour *chaque* programme, indiquez quelle est l'erreur et comment la corriger. Pour le programme 3, on rappelle qu'un *int* est codé sur 4 octets.

```cpp
#include <iostream>
using namespace std;

int main(void)
{
     int i = 2;
     int j = 3;
     float ratio, value;
     ratio = 1.0 * i / j;

     float result = value * ratio;
     cout << result << endl;
return 0;
}
```

Programme 1

```cpp
#include <iostream>
using namespace std;

int diff(int *a, int *b)
{  return b - a;  }

int main(void)
{
    int a[] = {1,4};
    cout << "b - a = " <<
            diff(a,a+1) << endl;
return 0;
}

// Affiche "b - a = 1"
//       et non "b - a = 3" !??
```

Programme 2

```cpp
#include <iostream>
using namespace std;
int exp(int N , int P)
{
    int RESULT=1;
    for(int i=0; i<P;i++)
        RESULT=RESULT*N;
    return RESULT;
}

int main(void)
{
    int nombre, result, puissance;
    nombre    = 15;
    puissance = 8;
    result=exp( nombre, puissance);
    cout << result << endl;
// Si nombre = 2  -> result = 256;
// Si nombre = 16 -> result = 0;
// Si nombre = 15 -> result = -
1732076671;
return 0 ;
}
```

*Programme 3 (*un *int* est codé sur 4 octets)

```cpp
#include <iostream>
using namespace std;

void echange(int a, int b)
{ int t=b;
  b=a;
  a=t;
}

int main(int argc, char ** argv)
{
    int a=10, b=20;
    cout << "a = " << a << endl
         << "b = " << b << endl;
    echange(a,b);

    cout << "a = " << a << endl
         << "b = " << b << endl;
return 0;
}
```

Programme 4

2) Modifier les classes B suivantes pour que les programmes compilent, s'exécutent et produisent un résultat cohérent**. Les fonctions 'main' et les classes A sont justes et ne doivent pas être modifiées**. Il n'y a qu'**une** erreur par programme.

```cpp
#include <iostream>
using namespace std;

class A {
  public:
    double Xa;
    A(const double &x):Xa(x) {}
};

class B : public A { };

int main()
{   B b;
    cout << b.Xa << endl;
    return 0;
}
```

Programme 1

```cpp
#include <iostream>
using namespace std;

class A {
  public:
    virtual void Print()=0;
};
class B : public A {
double Xb;
  public:
     B():Xb(1) {}
};

int main()
{   A* b = new B;
    b->Print();
    delete b;
    return 0;
}
```

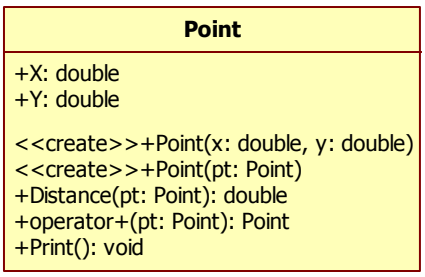
Programme 2

Exercice : Calculs sur des Polygones (45 minutes)

Le but de cet exercice est de réaliser une application capable de manipuler des polygones. Les traitements et manipulations seront dans des classes dédiées. Dans un premier temps, on s'intéresse aux constituants des polygones (les sommets, qui seront des points du plan) puis à la structure informatique de représentation des polygones (liste doublement chainée).

Partie : Class Point

Le diagramme UML de la classe Point est donné ci-après.



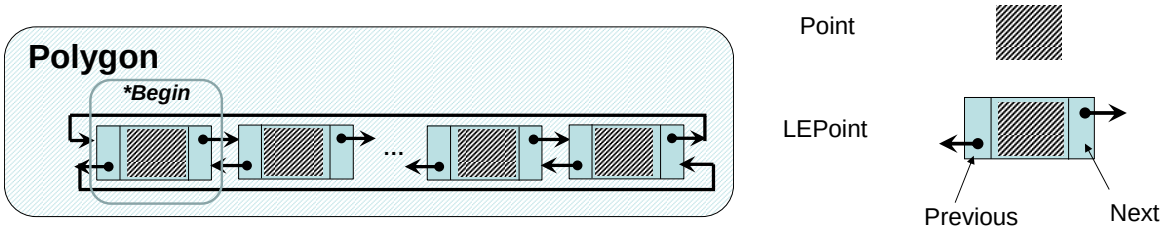
Moo.1 Combien de constructeurs sont présents dans cette classe ? Donner leurs rôles et les opérations qu'ils réalisent.

Moo.2 Donner la définition C++ de cette classe (le .h …).

Moo.3 Donner les implémentations des fonctions membres *Distance()* et *operator+()*.

Partie : Class Polygon

Un polygone sera une liste de points. Ces points représentent les sommets. Les côtés sont implicites : il existe un côté entre 2 points consécutifs dans la liste. La figure suivante illustre la représentation informatique des polygones.



La classe ***Point*** est celle écrite dans la partie précédente. Un objet ***Polygon*** contient ses sommets qui sont de type ***LEPoint***. La classe ***LEPoint*** (pour List Element Point) ajoute les fonctionnalités nécessaires à la classe ***Point*** pour pouvoir être utilisée dans une liste doublement chainée comme l'est la classe ***Polygon***.

La définition de la classe ***Polygon*** et les codes de quelques fonctions dont *Copy(), Print()* et *Clear()* sont donnés ci-après.

```
class Polygon
{
    LEPoint *Begin; // Pointe le premier element de la liste (le premier sommet)
    unsigned int NbPoints; // donne le nombre de sommets du polygone
public:
    Polygon(): NbPoints(0) {Begin = NULL;}
    ~Polygon() {  Clear(); }

    void Clear();
    unsigned int GetNbPoints() const {return NbPoints;}

    void AddPointsAfter(const Point &pt, LEPoint *pos=0);
    void AddPointsAfter(const double &x, const double &y, LEPoint * pos=0);

    Point RemovePoint(LEPoint * pos);

    bool IsEmpty() {return (NbPoints == 0);}
    LEPoint * GetBegin() const {return Begin;}

    void Print() const;
    void Copy(const Polygon &p);
};
```

```
void Polygon::Copy(const Polygon &p)
{    Clear();
    LEPoint *it_p = p.Begin;
    AddPointsAfter(*it_p, NULL);
    LEPoint *it_this = Begin;
    it_p = it_p->Next;
    for(unsigned int i=1;
                i<p.GetNbPoints(); i++)
        {
        AddPointsAfter(*it_p, it_this );
        it_p = it_p->Next;
        it_this = it_this->Next;
        }
}

void Polygon::Clear()
{    while( ! IsEmpty() )
    {
        LEPoint *tmp = Begin->Next;
        NbPoints--;
        delete Begin;
        Begin = tmp;
    }
}

void Polygon::Print() const
{    cout << "[" << NbPoints << "] = ";
    Begin->Print();
    LEPoint *tmp = Begin->Next;
    while( tmp != Begin )
        {
        tmp->Print();
        tmp = tmp->Next;
        }
    cout << endl;
}
```

Les méthodes *AddPointAfter()* insèrent un point **après** le sommet pointé par *pos*. Si *pos* est nul, l'insertion est faite avant *Begin* (avant le premier sommet).

Moo.4 Le destructeur de la classe ***Polygon*** est-il obligatoire ? Pourquoi ?

Moo.5 Modéliser les classes ***Polygon*** et ***LEPoint*** en faisant apparaitre les liens entre ces classes et la classe ***Point***. Justifier les liens entre ces classes.

Moo.6 On veut implémenter l'*operator=()* de la classe ***Polygon***. Donner la définition et l'implémentation C++ de cet opérateur.

Moo.7 Parmi les instructions suivantes, lesquelles sont fonctionnelles et compilables ? Justifier. Pour les méthodes fonctionnelles donner les modifications du polygone.

```
Point pt(4,3);
Polygon p1;
   p1.AddPointsAfter(1,1, 0);
   p1.AddPointsAfter(2,2);
   p1.AddPointsAfter(pt, p1.GetBegin() );
   p1.AddPointsAfter(Point(3,2), 0);
   p1.AddPointsAfter(LEPoint(3,3), 0);
```

Partie : Classe ComputePerimeter

La classe ***ComputePerimeter*** prend l'adresse d'un objet ***Polygon*** en entrée (via la méthode *SetInput(...)*) et stocke un double (*Output*) qui contiendra le résultat du calcul. Ainsi, lors de l'appel de la fonction *Update()*, le calcul du périmètre du polygone est fait et stocké dans *Output*. L'utilisateur récupère cette valeur par l'accesseur *GetOutput()*. Les champs *Input* et *Output* sont privés. Le code suivant illustre une utilisation de la classe ***ComputePerimeter***.

```
Polygon p2;
 // here some points are added as : p2.AddPointsAfter(0,0,0);
...
ComputePerimeter cp;
cp.SetInput(&p2);
cp.Update();
cout << "Perimeter = " << cp.GetOutput() << endl;
```

Moo.8 Donner le diagramme UML de la classe ***ComputePerimeter*** en explicitant les liens avec les autres classes du système.

Moo.9 Donner le code C++ de la fonction *Update()*.

Moo.10 On souhaite créer la classe ***ComputeArea*** se comportant comme ***ComputePerimeter*** mais qui calcule l'aire du polygone. Proposer une modélisation de cette classe et une architecture adaptée.

Fin.