EISTI	Cycle ingénieur 2ème année – GSI Examen de Programmation C++ Mohamed MAACHAOUI, Rémi VERNAY	
	Appareils électroniques et documents interdits	*Date :* **13 Mai 2016** *Durée de l'épreuve :* **2 heures** *Nombre de pages du sujet :* **9 pages**

Exercice 1 : Les tours de Hanoï (13 pts)

Le jeu des tours de Hanoï a été inventé par le mathématicien français Lucas sous le pseudonyme de Claus. Il se présente sous la forme d'un support en bois sur lequel sont plantées trois tiges *A*, *B* et *C* qui symbolisent les tours. Sur ces trois tiges peuvent être enfilés des disques de diamètres différents (8 dans la version originale, mais *N* de manière générale). Dans la configuration initiale (figure 1), les disques sont empilés par ordre de taille décroissante sur la tige *A*.

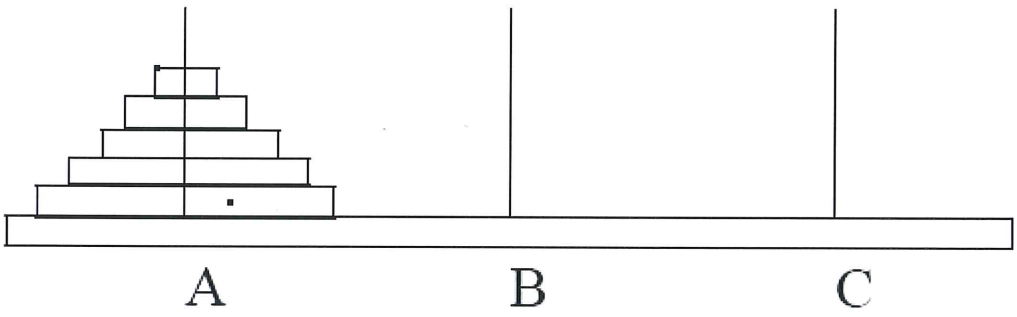

FIG. 1 – Configuration originale du jeu des tours de Hanoï

Le but du jeu est de transférer tous les disques de la tige *A* vers la tige *C* sachant que :

- un seul disque peut-être déplacé à la fois ;
- seul le disque en haut d'une tige (tour) peut être déplacé ;
- un disque ne peut jamais être posé sur un disque de taille inférieure à la sienne.

L'objectif de cet exercice est de proposer une modélisation objet de ce jeu.

1.1. Le diagramme de classes du jeu des tours de Hanoï est donné à la figure 2. Expliquer ce diagramme de classes.

1.2. Les disques. Écrire le code C++ de la classe Disque.

1.3. Les tours. Intéressons-nous maintenant à la classe Tour.

Remarque : Il est possible de regrouper vos réponses pour les questions suivantes.

1.3.1. Expliquer le sens des opérations de la classe Tour.

1.3.2. On veut conserver la généralité du jeu de Hanoï et ne pas limiter le nombre de disques à 8. Comment choisir les attributs et quelles sont les conséquences sachant que les disques seront stockés dans un tableau ?

1.3.3. On décide que l'affectation et la copie entre tours n'ont pas de sens. Comment faire pour le modéliser ?

1.3.4. Formaliser le comportement de la classe Tour en utilisant la programmation par contrat.

1.3.5. Écrire en C++ la classe Tour.

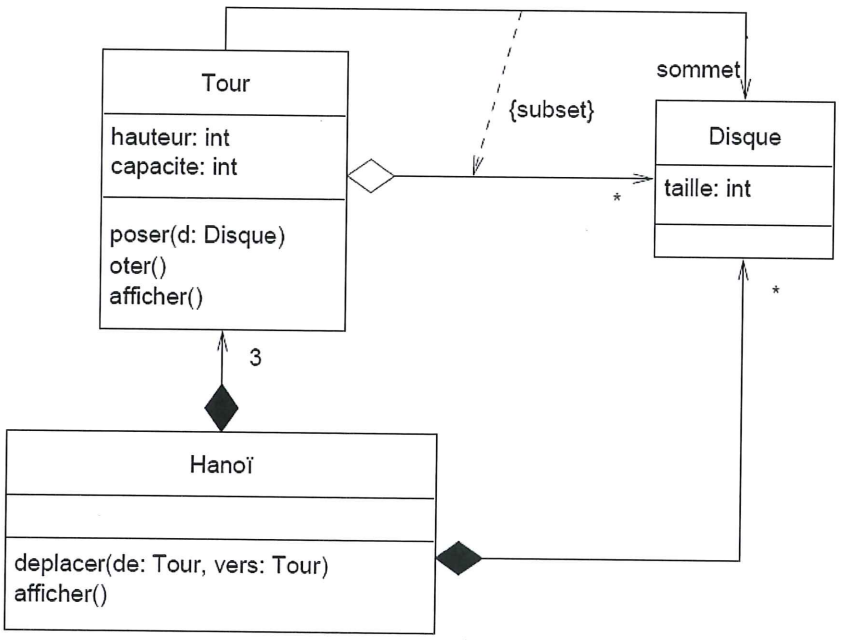

FIG. 2 – Le diagramme de classes du jeu des tours de Hanoï

L'appel à l'opérateur << affiche la taille des disques (du plus grand au plus petit). Si la tour n'est pas pleine, un tiret indiquer une position inoccupée. En voici quelques exemples :

```
(8) (7) (4) (1) (-) (-) (-) (-)

(10) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-)

(8) (7) (6) (5) (4) (3) (2) (1)
```

1.4.*Hanoï.* La déclaration de la classe Hanoi est fournie (listing 1). Donner son implantation.

Listing 1 : Le fichier Hanoi.h

```
1   // Objectif : Modéliser le jeu des tours de Hanoï
2   #ifndef Hanoi__H
3   #define Hanoi__H
4   #include "Tour.h"
5
6   class Hanoi {
7       public:
8           Hanoi(unsigned int n);
9               // Initialiser le jeu des tour de Hanoï avec n disques
10
11          Tour &tour( unsigned int numero);
12              // La tour de numéro i
13
14          void deplacer (Tour &de, Tour &vers);
15              // déplacer le disque le plus haut de la tour « de » vers
16              // la tour « vers ».
17
18      private :
19          Tour tour1 ;
20          Tour tour2 ;
21          Tour tour3 ;
22  };
23  std::ostream & operator << (std::ostream & out, const Hanoi &h);
24      // Afficher les trois tours
25  #endif
```

1.5.Application. Intéressons-nous maintenant à la résolution du problème des tours de Hanoï.

1.5.1. Écrire un programme qui résout le jeu des tours de Hanoï dans le cas où il n'y que deux disques ($N = 2$).

1.5.2. Écrire un programme qui résout le jeu des tours de Hanoï dans le cas général.

Indication : On pourra utiliser un raisonnement par récurrence et définir la fonction :

```
void deplacer (Hanoi &hanoi, int n, int a, int c, int b);
   // Déplacer n disques de la tour de numéro a vers la tour numéro c en
   // utilisant la tour numéro b.
```

Exercice 2 : Modélisation d'un segment (7 pts)

L'objectif de cet exercice est de réfléchir à une modélisation particulière des segments. Nous considérons les classes Point (listings 2 et 3) et Segment (listings 4 et 5).

2.1. Indiquer ce qui est affiché sur l'écran après l'exécution du programme du listing 6.

2.2. En déduire quelle est la relation entre le segment et ses points extrémités. Dessiner le diagramme de classes correspondant à cette application.

2.3. Expliquer pour quelle raison la longueur du segment n'est pas cohérente.

2.4. Quels commentaires pouvez-vous faire sur la manière dont cette classe est écrite ?

2.5. Nous souhaitons définir une solution pour que la longueur du segment soit toujours cohérente. Cette solution devra être générale, c'est-à-dire qu'elle devra pouvoir être appliquée à chaque fois qu'un objet d'une classe dépend des changements intervenants sur des objets d'une autre classe. En particulier, la solution proposée doit respecter les contraintes suivantes :

- la relation entre Segment et Point reste inchangée ;
- l'attribut longueur et la fonction longueur() de Segment restent inchangés.

2.5.1. Une première solution envisagée est de modifier la classe Point pour que chaque changement sur le point soit répercuté sur le segment. Indiquer les modifications à apporter.

Un point peut-il être utilisé pour construire plusieurs segments ? Ceci remet-il en cause votre solution ?

2.5.2. La solution précédente n'est pas complètement satisfaisante car elle nécessite que la classe Point connaisse la classe Segment et, plus généralement, toutes les classes qui vont dépendre de ses changements. Proposer une solution informelle et le diagramme de classes correspondant à une solution plus générale dans laquelle la classe Point ne connaît pas ces autres classes.

Listing 2 : Le fichier Point.h

```
1  // Objectif :
2  //      Définition d'un point dans un espace plan ( à deux dimmensions).
3  #ifndef Point__H
4  #define Point__H
5
6  class Point
7  {
8      public:
9          Point ( double vx, double vy);
10             // Constructeur à partir des coordonnées cartésiennes vx et vy.
11             //
12             // @post x() == vx;
13             // @post y() == vy;
14
15         double x() const;
16             // Abscisse du point
17
18         double y() const;
19             // Ordonnée du point
20
21         double distance ( const Point &autre ) const;
22             // Distance du point à un autre .
23
24         void translater ( double dx, double dy);
25             // Translater le point de dx suivant l ' axe des X
26             // et dy suivant l ' axe des Y
27             //
28             // @post x() == x() @pre + dx;
29             // @post y() == y() @pre + dy;
30
31         double _x; // Abscisse du point
32         double _y; // Ordonnée du point
33 };
34 std::ostream & operator << (std::ostream & out, const Point &p);
35     // Afficher le point
36 #endif
```

Listing 3 : Le fichier Point.cpp

```
1  // Objectif : Implantation de la classe Point
2
3  #include "Point.h"
4  #include <iostream>
5  #include <cmath>
6  #include <assert.h>
7
8  namespace {
9      double carre ( double x)
10         // le carré de x
11     {
12         return x * x;
13     }
14 }
15
16 Point :: Point (double x, double y)
17 {
18     _x = x;
19     _y = y;
20 }
21
22 double Point :: x() const
23 {
24     return _x;
25 }
26
27 double Point :: y() const
28 {
29     return _y;
30 }
31
32 void Point :: translater ( double dx, double dy)
33 {
34     // translater chaque coordonnée
35     _x = _x + dx;
36     _y = _y + dy;
37 }
38
39 double Point :: distance ( const Point &autre ) const
40 {
41     return sqrt ( carre(x() - autre.x()) + carre(y() - autre.y()));
42 }
43
44 std::ostream & operator << (std::ostream & out, const Point &p)
45 {
46     out << "(" << x() << "," << y() << ")";
47     return out;
48 }
```

Listing 4 : Le fichier Segment.h

```
1  // Objectif :
2  //      Définition d'un segment caractérisé par ses deux extrémités .
3
4  #ifndef Segment__H
5  #define Segment__H
6  #include "Point.h"
7
8  class Segment {
9      public:
10         Segment(Point &p1, Point &p2);
11             // Initialiser le segment à partir de ses deux extrémités p1 et p2
12             //
13             // @post e1() == p1 // e1 initialisé
14             // @post e2() == p2 // e2 initialisé
15             // @post &e1() != &p1; // e1 différent de p1
16             // @post &e2() != &p2; // e1 différent de p2
17
18         const Point & e1() const;
19             // une extrémité du segment
20
21         const Point & e2() const;
22             // l'autre extrémité du segment
23
24         double longueur() const;
25             // longueur du segment
26
27         void translater ( double dx, double dy);
28             // Translater le Segment de dx et dy suivant
29             // l'axe des X et l'axe des Y
30
31         Point * _e1, * _e2; // les deux extrémités du segment
32         double _longueur; // longueur du segment
33 };
34 std::ostream & operator << (std::ostream & out, const Segment &s);
35     // Afficher le segment
36 #endif
```

Listing 5 : Le fichier Segment.cpp

```
1  // Objectif : Implantation de la classe Segment
2
3  #include "Segment.h"
4  #include <iostream>
5
6  Segment :: Segment(Point &p1, Point &p2)
7  {
8      _e1 = &p1;
9      _e2 = &p2;
10     _longueur = e1(). distance ( e2());
11 }
12
13 const Point & Segment :: e1() const
14 {
15     return *_e1;
16 }
17
18 const Point & Segment :: e2() const
19 {
20     return *_e2;
21 }
22
23 double Segment :: longueur() const
24 {
25     return _longueur;
26 }
27
28 void Segment :: translater ( double dx, double dy)
29 {
30     _e1->translater (dx, dy);
31     _e2->translater (dx, dy);
32 }
33
34 void Segment :: afficher () const
35 {
36     cout « ' [' ;
37     e1().afficher ();
38     cout « " ; ";
39     e2().afficher ();
40     cout « ' ]' ;
41 }
42 std::ostream & operator << (std::ostream & out, const Segment &s)
43 {
44     out « ' [' « e1() « " ; " « e2() « ' ]' ;
45     return out ;
46 }
```

Listing 6 : Le fichier testSegment.cpp

```
1   // Objectif : Implantation de la classe testSegment
2
3   #include "Point.h"
4   #include "Segment.h"
5   #include <iostream>
6   #include <stdlib.h>
7
8   int main()
9   {
10      Point * p1 = new Point(0, 0);
11      Point p2(5, 0);
12      Segment s(*p1, p2);
13
14      cout « "p2 = " « p2 « endl;
15      cout « "s = " « s « endl;
16      cout « "longueur de s = " « s. longueur() « endl;
17
18      p2.translater (- 2, 0);
19
20      cout « "p2 = " « p2 « endl;
21      cout « "s = " « s « endl;
22      cout « "longueur de s = " « s.longueur() « endl;
23
24      // Afficher l'état de la mémoire ICI
25
26      return EXIT_SUCCESS;
27  }
```