Université de Nice-Sophia Antipolis — POLYTECH
ELEC4 — 2020–2021

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

On souhaite représenter des triangles dans le plan cartésien. Pour cela, on souhaite d'abord définir la classe `Point` suivante :

```
class Point {
private:
    double x,y;
public:
    Point(const double a=0, const double b=0) : x(a), y(b){}
    double getX() const;
    double getY() const;
    void setX(const double x);
    void setY(const double y);
    double distance(const Point &p) const;
    bool égal(const Point &p) const;
    std::string toString() const;
    friend std::ostream& operator<<(std::ostream &f,const Point &p);
};
```

► 1. Écrivez les méthodes `distance`, `égal` et **`operator<<`** comme elles doivent être écrites dans le fichier `Point.cpp`.

Programmation des méthodes dans le fichier `Point.cpp`

```
/*
 * Rôle : renvoie la distance entre le Point courant et le Point p
 */
double Point::distance(const Point &p) const {
  double dx = p.getX()-this->getX();
  double dy = p.getY()-this->getY();
  return sqrt(dx*dx+dy*dy);
}

/*
 * Rôle : renvoie la représentation sous formed de std:string
 *        du Point courant
 */
std::string Point::toString() const {
  return"("+std::to_string(this->getX())
    +","+std::to_string(this->getY())+")";
}

/*
 * Rôle :surcharge de l'opérateur << pour écrire le Point p
 * sur l'ostream f
*/
std::ostream &operator<<(std::ostream &f,const Point &p) {
  return f << p.toString();
}

/*
 * Rôle : teste si le Point courant et le Point p
 *        sont égaux à un epsilon près .
 */
bool Point::égal(const Point &p) const {
  const double eps=1e-10; //précision des calculs
  return fabs(this->getX()-p.getX())<eps && fabs(this->getY()-p.getY())<eps;
}
```

.....

► 2. On veut maintenant définir la classe `Triangle`. Écrivez cette classe qui comporte :
- les 3 sommets du triangle sous forme d'un tableau de 3 pointeurs sur `Point` ;
- un constructeur qui prend en paramètres 3 pointeurs sur `Point`. Vous vérifierez que les 3 sommets sont distincts ;
- une méthode qui renvoie le périmètre du triangle courant ;
- la surcharge de l'opérateur `<<`.

Vous programmerez les méthodes demandées.

Déclaration de la classe `Triangle` dans le le fichier `Triangle.hpp`.

```
class Triangle {
private:
  Point **sommets;
public:
  // le constructeur
  Triangle(Point *a, Point *b, Point *c) : sommets(new Point*[3]) {
    assert(!a->égal(*b) && !a->égal(*c) && !b->égal(*c));
    // dupliquer les Points.
    // Note: le constructeur de copie( par défaut)
    // de Point est appliqué
    sommets[0] = new Point(*a);
    sommets[1] = new Point(*b);
    sommets[2] = new Point(*c);
  }

  double périmètre() const;
  std::string toString() const;
  friend std::ostream &operator<<(std::ostream &f,const Triangle &t);
```

Programmation des méthodes dans le le fichier `Triangle.cpp`.

```
/*
 * Rôle : renvoie la représentation sous forme de std::string
 * du Triangle courant
 */
std::string Triangle::toString() const {
  return "[" +
    this->sommets[0]->toString() + "," +
    this->sommets[1]->toString() + "," +
    this->sommets[2]->toString() +
    "]";
```

```
}

/*
 * Rôle : renvoie le périmètre du triangle courant
 */
double Triangle::périmètre() const {
  return this->sommets[0]->distance(*sommets[1])
    +this->sommets[1]->distance(*sommets[2])+
    this->sommets[2]->distance(*sommets[0]);
}

/*
 * Rôle : écrit le Triangle t sur le std::ostream f
 */
std::ostream &operator<<(std::ostream &f, const Triangle &t) {
  return f << t.toString();
}
```

.....

► 3. Est-ce qu'il est nécessaire d'écrire un destructeur et un constructeur de copie pour la classe `Triangle` ? Si oui, expliquez pourquoi et programmez-les.

Il est nécessaire d'écrire un destructeur pour libérer explicitement la mémoire occupée par le tableau et les 3 points du triangle alloués dynamiquement. De même, le constructeur de copie est nécessaire pour la copie de même tableau et ces mêmes points.

```
// constructeur de copie
Triangle(const Triangle &t ) {
  // allouer le tableau sommets
  this->sommets = new Point*[3];
  // dupliquer les 3 sommets
  for (int i=0; i<3; i ++)
     this->sommets[i] = new Point (t.sommets[i]->getX(), t.sommets[i]->getY());
}

//destructeur
~Triangle() {
   //désallouer les 3 sommets
   for(int i=0; i<3; i++) delete this->sommets[i];
}
```

.....

► 4. Écrivez la fonction `main` qui :
— déclare un triangle `t` formé des 3 points $(1,1)$, $(2,2)$ et $(0,2)$;
— écrit sur la sortie standard le triangle et son périmètre ;
L'exécution de cette fonction `main` pourra produire :

```
t = [(1.000000,1.000000),(2.000000,2.000000),(0.000000,2.000000)]
périmètre de t : 4.82843
```

```
#include <cstdlib>
#include <iostream>
#include "Triangle.hpp"

int main() {

  Triangle t= Triangle(new Point(1,1), new Point(2,2), new Point(0,2));
  std::cout << "t=" << t <<std::endl;
  std::cout << "périmètre de t: "<< t.périmètre() <<std::endl;

  return EXIT_SUCCESS;
}
```