Université de Nice-Sophia Antipolis — POLYTECH
ELEC4 — 2018–2019

Examen de Programmation C++

Durée : 0h30
Aucun document autorisé
Mobiles interdits — Nom : — Prénom :

On possède la classe suivante pour représenter les points du plan cartésien.

```
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;
  friend std::ostream&operator<<(std::ostream &f, const Point &p);
};
```

Notez que l'implémentation de ces méthodes ne vous est pas donnée, car inutile. Le nom de chaque méthode est suffisamment significatif.

On souhaite représenter des polygones dans le plan cartésien. On rappelle qu'un polygone est constitué d'une suite finie de points du plan appelés sommets et de segments, appelés côtés, reliant les couples de sommets consécutifs ainsi que d'un segment reliant le premier et le dernier point.

▶ 1. Écrivez en C++ la classe `Polygone` avec :

- une variable privée `sommets` type vecteur de `Point` ;
- un constructeur pour initialiser l'objet courant ;
- l'accesseur `getSommet` qui renvoie le ième sommet du polygône courant ;
- le mutateur `setSommet` qui modifie le ième sommet du polygone courant ;
- la fonction `périmètre` qui renvoie le périmètre du polygone courant ;
- la surcharge de l'opérateur `<<` pour écrire sur un `ostream` un polygone au format $[(x_1, y_1)(x_2, y_2) \dots (x_i, y_i) \dots (x_n, y_n)]$.

```
// Polygone.hpp
#pragma once

#include <vector>
#include <iostream>
#include "Point.hpp"
/*
 * La classe Polygone représente des polygones à n sommets (n>=1)
 */
class Polygone {

private:
  std::vector<Point> sommets;
public:
  // les constructeurs
  Polygone(const std::vector<Point> &pts) : sommets(pts) {}
  // Rôle : renvoie le nombre de sommets du polygone courant
  int nbSommets() const;
  // Rôle : renvoie le ième sommet du Polygone courant
  // 0 <= i < nbSommets()
  const Point& getSommet(const int i) const;
  // Rôle : change le ième sommet du Polygone courant
  // 0 <= i < nbSommets()
  void setSommet(const int i, const Point &p);
  // Rôle : renvoie du périmètre du Polygone courant
  double perimetre() const;
  // Rôle : écrit le Polygone p sur le flot f
  friend std::ostream&operator<<(std::ostream &f, const Polygone &p);
};
```

```
// Polygone.cpp
#include <iostream>
#include <cassert>
#include "Polygone.hpp"

// Rôle : renvoie le nombre de sommets du polygone courant
int Polygone::nbSommets() const {
  return this->sommets.size();
}

// Rôle : renvoie le ième sommet du Polygone courant
// 0 <= i < nbSommets()
Point Polygone::getSommet(const int i) const {
  assert(i>=0 && i<this->sommets.size());
  return this->sommets[i];
}

// Rôle : change le ième sommet du Polygone courant
// 0 <= i < nbSommets()
void Polygone::setSommet(const int i, const Point &p) {
  assert(i>=0 && i<this->sommets.size());
  this->sommets[i] = p;
}

// Rôle : renvoie du périmètre du Polygone courant
double Polygone::perimetre() const {
  double p = 0.0;
  int nbS = this->sommets.size();
  for (int i=1; i<nbS; i++)
    p+=this->sommets[i-1].distance(this->sommets[i]);
```

```
  if (nbS>=3)
    // polygone fermé => ajouter la longueur du dernier coté
    p+=this->sommets[0].distance(this->sommets[nbS-1]);
  //
  return p;
}

// Rôle : écrit le Polygone p sur le flot f
std::ostream& operator<< (std::ostream& f, const Polygone &p)
{
  f << "[ ";
  for (auto x : p.sommets) f << x << " ";
  return f << "]";
}
```

.....

▶ 2. Écrivez la fonction `main` qui :

1. déclare et construit un triangle et un pentagone. Le premier polygone sera alloué dynamiquement, mais pas le second.
2. puis écrit sur la sortie standard ces deux polygones et leurs périmètres respectifs.

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include "Point.hpp"
#include "Polygone.hpp"


int main() {

  std::vector<Point> pts3 = {Point(1,1), Point(2,2), Point(3,3)};
  Polygone *triangle = new Polygone(pts3);
  std::cout << *triangle << " - périmètre = ";
  std::cout << triangle->perimetre() << std::endl;

  std::vector<Point> pts5 = {
    Point(1,1), Point(2,2), Point(3,3), Point(4,4), Point(5,5)
  };
  Polygone pentagone(pts5);
  std::cout <<  pentagone << " - périmètre = ";
  std::cout << pentagone.perimetre() << std::endl;

  std::cout << Polygone({Point(0,0), Point(1,1)}) << std::endl;
  std::cout << Polygone({Point(0,0), Point(0,1)}).perimetre() << std::endl;
  std::cout << Polygone({Point(0,0), Point(0,1), Point(1,1)}).perimetre()
            << std::endl;
  std::cout << Polygone({Point(1,1)}).perimetre() << std::endl;

  return EXIT_SUCCESS;
}
```


.....