Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Программирование

Лабораторная работа №1

ООП. Классы.

Выполнил студент группы № M3109
Гумбатов Владислав Юрьевич
Подпись:

Санкт-Петербург
2021

Задача

Целью лабораторной работы является проектирование и реализация следующих классов:

1) Точка
2) Ломанная
3) Замкнутая ломанная
4) Многоугольник
5) Трапеция
6) Правильный многоугольник

Для каждого из классов необходимо реализовать следующие методы:

1) Конструкторы
2) Конструкторы копирования
3) Оператор присваивания
4) Расчет периметра
5) Расчет площади
6) Приватные и публичные методы по усмотрению

Входные данные:

Входными данными в программе являются параметры, для конструкторов объектов классов.

Выходные данные:

В качестве выходных данных программа выводит результаты выполнения операций методов классов.

Код:

```
#include <iostream>
#include <list>
#include <math.h>
#include <cmath>

const double PI = 3.14159265358979323846264338327795;

class Point {

public:
    Point(double fCoord = 0, double sCoord = 0) {
        this->x = fCoord;
        this->y = sCoord;
    }
    Point(const Point& other) {
        this->x = other.x;
        this->y = other.y;
    }
    Point& operator= (const Point& point) {
        x = point.x;
        y = point.y;
        return *this;
    }
    double getX() const {
        return x;
    }
    double getY() const {
        return y;
    }
    void print() const {
        std::cout <<"x = " << this->getX() << "  " << "y = " << this->getY() << std::endl;
    }
private:
    double x;
    double y;

};

class Polyline {
public:

    Polyline(Point a, Point b, Point c) {

        this->listOfPoints.push_back(a);
        this->listOfPoints.push_back(b);
        this->listOfPoints.push_back(c);


    }

    void AddPoint(Point x) {

        this->listOfPoints.push back(x);
```

```
 void printPolyLine() {

     int len = listOfPoints.size();

     this->iter = this->listOfPoints.begin();


     for (int i = 0; i < len; i++) {

         this->iter->print();

         iter++;
     }


 }

 double lenOfPolyLine() {

     int lenOfList = listOfPoints.size();

     double a; double b; double c; double d;

     double len = 0;

     this->iter = this->listOfPoints.begin();

     for (int i = 0; i < lenOfList - 1; i++) {

         a = this->iter->getX();
         b = this->iter->getY();
         iter++;
         c = this->iter->getX();
         d = this->iter->getY();

         len = sqrt(pow((c - a), 2) + pow((d - b), 2)) + len;


     }

     return len;

 }

 void printLenOfPolyLine() {

     double len;

     len = lenOfPolyLine();

     std::cout << len << std::endl;
```

```
    Polyline(const Polyline& other) {

        this->listOfPoints = other.listOfPoints;
    }


    Polyline& operator = (const Polyline& polyline) {

        this->listOfPoints = polyline.listOfPoints;
        return *this;

    }


protected:

    std::list<Point> listOfPoints;
    std::list<Point>::iterator iter;
};


class LockedPolyLine :public Polyline {
public:

    LockedPolyLine(Point a, Point b, Point c) :Polyline(a, b, c) {
    }

    LockedPolyLine(const LockedPolyLine& other) :Polyline(other) {

    }

    LockedPolyLine& operator = (const LockedPolyLine& locledPolyLine) {

        this->listOfPoints = locledPolyLine.listOfPoints;

    }

    void AddPoint(Point x) {

        Polyline::AddPoint(x);

    }

    double lenOfLockedPolyLine() {

        double lenA = 0;

        //Point one; Point two;


        this->iter = this->listOfPoints.begin();
          double a = this->listOfPoints.begin()->getX();


          double b = this->listOfPoints.begin()->getY();


          double c = this->listOfPoints.back().getX();


          double d = this->listOfPoints.back().getY();


          lenA = sqrt((pow((c - a), 2) + pow((d - b), 2)));

          //one = *this->iter;

          //this->iter = this->listOfPoints.end();

          //two = *this->iter;

          //lenA = sqrt((pow(two.getX() - one.getX(), 2)) + (pow(two.getX() - one.getY(), 2)));

              /*lenA = sqrt((pow((*(this->Polyline::listOfPoints.end())).getX() - (*(this->Polyline::listOfPoints.begin())).getX(), 2)
                  + pow((*(this->Polyline::listOfPoints.end())).getY() - ((*(this->Polyline::listOfPoints.begin()))).getY(), 2)));*/


              return this->lenOfPolyLine() + lenA;

      }

      void printLenOfLockedPolyLine() {

          double len = lenOfLockedPolyLine();
          std::cout << len << std::endl;

      }



  private:

  };


  class PolyAngle:public LockedPolyLine {

  public:

      PolyAngle(Point a, Point b, Point c):LockedPolyLine(a, b, c) {

      }
```

```
PolyAngle(const LockedPolyLine& other):LockedPolyLine(other) {

}



PolyAngle& operator = (const PolyAngle& polyAngle) {

    this->listOfPoints = polyAngle.listOfPoints;
}


double lenOfPolyAngle() {

    double len = 0 ;


    //len = sqrt((pow((*(listOfPoints.end())).getX() - (*listOfPoints.begin()).getX(), 2)
    //  + pow((*(listOfPoints.end())).getY() - ((*(listOfPoints.begin()))).getY(), 2)));

    //std::cout << lenOfLockedPolyLine() + len << std::endl;

    return LockedPolyLine::lenOfLockedPolyLine() + len;

}



void addPoint(Point x) {

    Polyline::AddPoint(x);

}

double findArea() {

    double a, b, c, d;

    double firstPass = 0; double secondPass = 0;

    int areaOfPolyAngle;

    int tempVar;

    int lenOfList = listOfPoints.size();

    iter = this->listOfPoints.begin();

    for (int i = 0; i < lenOfList - 1; i++) {


        a = this->iter->getX();

        c = this->iter->getY();

        firstPass = (a * c) + firstPass;


    }

    Point al = this->listOfPoints.back();

    Point af = this->listOfPoints.front();


    double x = al.getX();

    double y = af.getY();

    firstPass = firstPass + (x * y);

    iter = this->listOfPoints.begin();

    for (int i = 0; i < lenOfList - 1; i++) {


        b = this->iter->getY();
        iter++;
        d = this->iter->getX();

        secondPass = (b * d) + secondPass;

    }
    double y2 = al.getY();

    double x2 = af.getX();

    secondPass = secondPass + (y2 * x2);

    tempVar = firstPass - secondPass;

    if (tempVar < 1) {


        tempVar = tempVar * -1;
    }

    areaOfPolyAngle = (tempVar) * 0.5;


    std::cout << areaOfPolyAngle << std::endl;


    return areaOfPolyAngle;
```

```
private:

};


class TriForce:PolyAngle {

public:

    TriForce(Point a, Point b, Point c):PolyAngle(a,b,c) {

        check();

    }

    TriForce(const TriForce& other):PolyAngle(other) {

    }

    TriForce& operator = (const TriForce& triforce) {

        this->listOfPoints = triforce.listOfPoints;
    }

    double lenOfTriForce() {

        return PolyAngle::lenOfPolyAngle();
    }


    double areaOfTriangle() {


        double area = 0;

        double len = listOfPoints.size();

        double a, b, c;

        double fcx, fcy, scx, scy;

        this->iter = this->listOfPoints.begin();

        fcx = this->iter->getX();
        fcy = this->iter->getY();
        iter++;
        scx = this->iter->getX();
        scy = this->iter->getY();

        a = sqrt(pow((scx - fcx), 2) + pow((scy - fcy), 2));

        fcx = scx;
        fcy = scy;

        this->iter = this->listOfPoints.begin();

        scx = this->iter->getX();
        scy = this->iter->getY();

        c = sqrt(pow((scx - fcx), 2) + pow((scy - fcy), 2));


        double p = (a + b + c) * 0.5;


        area = sqrt(p*(p-a)*(p-b)*(p-c));


        std::cout << area << std::endl;

        return area;
    }

    void check() {

        double len = listOfPoints.size();

        double fcx, fcy, scx, scy;

        this->iter = this->listOfPoints.begin();

        fcx = this->iter->getX();
        fcy = this->iter->getY();
        iter++;
        scx = this->iter->getX();
        scy = this->iter->getY();

        if (fcx == scx and fcy == scy) {

            std::cout << "Incorrect triangle" << std::endl;
            exit(1);

        }

        iter++;
        scx = this->iter->getX();
        scy = this->iter->getY();
        if (fcx == scx and fcy == scy) {

            std::cout << "Incorrect triangle" << std::endl;
            exit(1);

        }
```

```
        this->iter = this->listOfPoints.begin();

        iter++;

        fcx = this->iter->getX();
        fcy = this->iter->getY();
        iter++;
        scx = this->iter->getX();
        scy = this->iter->getY();

        if (fcx == scx and fcy == scy) {

            std::cout << "Incorrect triangle" << std::endl;
            exit(1);

        }

        std::cout << "Nice Triangle" << std::endl;
    }



private:
};


class Trapezium:PolyAngle {
public:
    Trapezium(Point a, Point b, Point c, Point d) :PolyAngle(a, b, c) {

        PolyAngle::AddPoint(d);

        check();



    }

    void check() {

        int len = listOfPoints.size();

        double a; double b; double c; double d;

        double tan1; double tan2; double tan3; double tan4;

        this->iter = this->listOfPoints.begin();

        for (int i = 0; i < len-1 ; i++) {

            a = this->iter->getX();
            b = this->iter->getY();
            iter++;
            c = this->iter->getX();
            d = this->iter->getY();

            if (i == 0) {
                tan1 = (d - b) / (c - a);
            }
            if (i == 1) {
                tan2 = (d - b) / (c - a);
            }
            if (i == 2) {
                tan3 = (d - b) / (c - a);
            }

        }
        Point al = this->listOfPoints.back();

        Point af = this->listOfPoints.front();


        double x = al.getX();
        double y = af.getY();

        double y2 = al.getY();

        double x2 = af.getX();

        tan4 = (y - y2) / (x2 - x);

        iter = listOfPoints.begin();

        if (tan1 == tan2 or tan1 == tan3 or tan1 == tan4 or tan2 == tan3 or tan2 == tan4 or tan3 == tan4) {

            std::cout << "Nice Trapezium" << std::endl;

        }
        else {
            std::cout << "Incorrect Trapezium" << std::endl;
            exit(1);
        }

    }

    Trapezium (const Trapezium& other):PolyAngle(other) {

    }

    Trapezium& operator = (const Trapezium& trapezium) {

        this->listOfPoints = trapezium.listOfPoints;
    }
```

```
    double lenOfTrapezium() {

        return PolyAngle::lenOfPolyAngle();
    }

    double areaOfTrapezium() {

        return PolyAngle::findArea();

    }


private:
};

class CoolPolyAngle:PolyAngle {

public:

    CoolPolyAngle (Point a, Point b, Point c, Point d) : PolyAngle(a, b, c) {

        PolyAngle::addPoint(d);

        check();

    }


    CoolPolyAngle(const CoolPolyAngle& other) :PolyAngle(other) {

    }

    CoolPolyAngle& operator = (const CoolPolyAngle& coolpolyange) {

        this->listOfPoints = coolpolyange.listOfPoints;
    }

    double lenOfCoolPolyAngle() {

        return PolyAngle::lenOfPolyAngle();
    }

    void check() {

        int len = listOfPoints.size();

        iter = this->listOfPoints.begin();

        double a; double b; double c; double d;

        double section = 0;

        a = this->iter->getX();

        b = this->iter->getY();

        iter++;

        c = this->iter->getX();

        d = this->iter->getY();

        section = sqrt(pow((c - a), 2) + pow((d - b), 2));

        double perimeter = lenOfCoolPolyAngle();

        double perimeter2 = section * len;

        if (perimeter == perimeter2) {

            std::cout << "Nice CoolPolyAngle" << std::endl;

        }
        else {
            std::cout << "Incorrect CoolPolyAngle" << std::endl;
            exit(1);
        }

    }


    void addPoint(Point x) {

        Polyline::AddPoint(x);

        check();
    }

    void areaOfCoolPolyAngle() {

        int len = listOfPoints.size();

        iter = this->listOfPoints.begin();

        double a; double b; double c; double d;

        double section = 0;

        a = this->iter->getX();
```

```
        b = this->iter->getY();
        iter++;
        c = this->iter->getX();
        d = this->iter->getY();
        section = sqrt(pow((c - a), 2) + pow((d - b), 2));

        double temp = 180 / len;
        double temp2 = ((temp*PI)/180);
        double tg = tan(temp2);
        double area = ( (len * pow(section, 2) ) /( 4 * tg));
        std::cout << area << std::endl;
    }


private:
};


int main()
{
    std::cout << "Point:" << std::endl;
    Point a(2, 4);
    a.print();
    std::cout << "_____________________" << std::endl;
    std::cout << "Polyline:" << std::endl << std::endl;
    Point b(3, -8);
    Point c(1, 2);
    Polyline line(a, b, c);
    std::cout << "Point in line: " << std::endl << std::endl;
    line.printPolyLine();
    std::cout << std::endl;
    std::cout << "Perimeter of line: " << std::endl;
    line.printLenOfPolyLine();
    std::cout << "_____________________" << std::endl;
    std::cout << "Locked Polyline:" << std::endl << std::endl;
    LockedPolyLine locked(a, b, c);
    std::cout << "Perimeter of line: " << std::endl;
    locked.printLenOfLockedPolyLine();
    std::cout << std::endl;
    std::cout << "_____________________" << std::endl;
    std::cout << "Polyangle:" << std::endl << std::endl;
    PolyAngle polyangle(a, b, c);
    std::cout << "Perimeter of polyangle: " << std::endl;
    std::cout << polyangle.lenOfPolyAngle() << std::endl;
    std::cout << std::endl;
    std::cout << "Area of polyangle: " << std::endl;
    polyangle.findArea();
    std::cout << "_____________________" << std::endl;
    std::cout << "Triangle:" << std::endl << std::endl;
    TriForce triangle(a, b, c);
    std::cout << std::endl;
    std::cout << "Perimeter of triangle: " << std::endl;
    std::cout << triangle.lenOfTriForce() << std::endl;
    std::cout << std::endl;
    std::cout << "Area of triangle: " << std::endl;
    triangle.areaOfTriangle();
```

```
triangle.areaOfTriangle();

std::cout << "_____________________" << std::endl;

std::cout << "Trapezium" << std::endl << std::endl;

Point d(9, 4);

Point fp(2, 2);
Point sp(7, 2);
Point tp(6, 4);
Point fop(4, 4);

Trapezium trapezium(fp, sp, tp, fop);

std::cout << std::endl;

std::cout << "Perimeter of trapezium: " << std::endl;


std::cout<<trapezium.lenOfTrapezium()<<std::endl;

std::cout << std::endl;

std::cout << "Area of trapezium: " << std::endl;


trapezium.areaOfTrapezium();

std::cout << std::endl;

std::cout << "_____________________" << std::endl;

std::cout << "Right Polyangle" << std::endl;

std::cout << std::endl;

Point r1(0, 0);
Point r2(0, 5);
Point r3(5, 5);
Point r4(5, 0);

CoolPolyAngle cpa(r1,r2,r3,r4);

std::cout << std::endl;

std::cout << "Perimeter of right polyangle: " << std::endl;

std::cout <<cpa.lenOfCoolPolyAngle() << std::endl;

std::cout << std::endl;

std::cout << "Area of right polyangle: " << std::endl;
```

Вывод:

Программа успешно работает. Цель лабораторной работы выполнена.