**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Лабораторная работа №7**

**«ООП. Простые классы»**

**по предмету «Алгоритмизация и программирование»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2023 г.**

1. **Общая постановка задачи**

## ***Задание 1.***

Для работы с точками на плоскости создать класс Point, содержащий:

1. *закрытые данные* – поля - координаты точки (x\_, y\_) - числа с плавающей точкой
2. *открытые данные* –

* конструктор без параметров, создающий точку (0.0,0.0);
* конструктор с параметрами, создающий точку с заданными координатами;
* конструктор копирования;
* деструктор;
* методы:
* setX, setY – для изменения значений закрытых полей (x\_, y\_);
* getX, getY – для показа значений закрытых полей (x\_, y\_);
* isEqual, проверяющий равны ли точки;
* getDistance, вычисляющий расстояние между двумя точками;
* move, перемещающий точку на K позиций по каждой координате.

*Написать функции:*

* isPointInCircle, проверяющую, попадает ли точка в круг, заданного радиуса (центр – (0,0), радиус – параметр, граница включается);
* input – для ввода закрытых полей (x\_, y\_) из потока cin;
* output – для вывода закрытых полей (x\_, y\_) в поток cout;

Объявление класса разместить в файле Point.h

Описание - в файле Point.cpp

Создать main для тестирования класса Point и функций.

Из потока cin ввести координаты точек (признак конца последовательности – точка с координатами (0,0).

Определить координаты точки, ближайшей к началу координат, и выведите их в поток cout.

## ***Задание 2.***

Для работы с треугольниками на плоскости создать класс Triangle, содержащий

1. *закрытые данные* – поля – объекты класса Point a\_, b\_, c\_
2. *открытые*:

* конструктор без параметров, создающий объект вырожденный треугольник с a\_, b\_, c\_, равными (0,0);
* конструктор с параметрами, создающий объект с заданными координатами вершин;
* конструктор копирования;
* деструктор;
* методы:
* setA, setB, setC – для изменения вершин;
* getA, getB, getC – для показа вершин;
* isTriangle, проверяющий, является ли a\_, b\_, c\_ - треугольником;
* move, перемещающую треугольник на K позиций;
* getPerimeter, определяющий периметр треугольника;
* isEqual, проверяющий равны ли треугольники (равны, если совпадают координаты всех вершин)

*Написать функции:*

* isEqualSquare, проверяющую равны площади треугольников;
* output – для выхода координат вершин в поток cout;

Объявление класса разместить в файле Triangle.h.

Описание в файле - Triangle.cpp.

Создать main для тестирования класса Triangle и функций.

1. **Детальные требования и тест план**

***1. Все переменные, вводимые с консоли (значения Х и Y точек, расстояние, на которое будет перемещена точка или треугольник, радиус окружности), должны быть заданы дробными числам:***

* Если переменная принимает нечисловое значение, то сообщение: «Некорректный ввод данных», аварийное завершение программы

***2. Для нахождения периметра заданные точки должны образовывать треугольник:***

* Если заданные точки не образуют треугольник (метод isTriangle вернул ложное значение), то сообщение «Фигура не является треугольником», аварийное завершение работы программы.

***3. Для сравнения площадей обе заданные фигуры должны являться треугольниками:***

* Если заданные точки не образуют треугольник хотя бы для одной из переданных в качестве параметров функции фигур (метод isTriangle вернул ложное значение), то сообщение «Фигура не является треугольником», аварийное завершение работы программы.

***Таблица с детальными требованиями и тест планом***

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | Детальные требования | Данные | Ожидаемый результат |
| 1. ***Вводимые с консоли переменные должны быть заданы корректно*** | | | |
| 1. Переменная - число | Если переменная принимает нечисловое значение, то сообщение:  «Некорректный ввод данных», аварийное завершение программы | rtryu | Сообщение:  «Некорректный ввод данных»  Аварийное завершение программы. |
| 2. Переменная задана корректно | Если переменная задана корректно, то продолжение работы программы | 0.5  0  7 | Продолжение работы программы |
| 1. ***При нахождение периметра фигура должна быть треугольником*** | | | |
| 1. Заданные точки не образуют треугольник | Если заданные точки не образуют треугольник (метод isTriangle вернул ложное значение), то сообщение «Фигура не является треугольником», аварийное завершение работы программы. |  | Сообщение: «Фигура не является треугольником»  Аварийное завершение работы программы. |
| 2. Заданные точки образуют треугольник | Если заданные точки образуют треугольник (метод isTriangle вернул положительное значение), то определение периметра треугольника. Продолжение работы программы. |  | Продолжение работы программы. |
| 1. ***При сравнении площадей обе фигуры должны являться треугольниками*** | | | |
| 1. Хотя бы 1 из заданных фигур не является треугольником | Если заданные точки не образуют треугольник хотя бы для одной из переданных в качестве параметров функции фигур (метод isTriangle вернул ложное значение), то сообщение «Фигура не является треугольником», аварийное завершение работы программы. |  | Сообщение: «Фигура не является треугольником»  Аварийное завершение работы программы. |
| 2. Если обе фигуры являются треугольниками | Если точки обеих фигур образуют треугольники, то сообщение о том, совпадают ли их площади или нет. Продолжение работы программы. |  | Продолжение работы программы. |

1. **Программа**

***Main.cpp***

#include"Point.h"

#include"Triangle.h"

int main()

{

setlocale(LC\_ALL, "ru");

try {

//тестироание класса Point

Point A; // вызов конструктора без параметров

std::cout << "A = ";

output(A);

Point B (3.5, 5); // вызов конструктора с параметрами

std::cout << "B = ";

output(B);

Point C(A); // вызов конструктора копирования

std::cout << "C = ";

output(C);

std::cout << "Расстояние от А до В = " << A.getDistance(B) << "\nРасстояние от А до С = " << A.getDistance(C) << "\n";

A.isEqual(B) ? std::cout << "Точки A и B совпадают\n" : std::cout << "Точки A и B не совпадают\n";

std::string value = "";

std::cout << "Растояние, на которое будет перемещена точка = \t";

std::cin >> value;

if (checkInput(value)) A.move(std::stod(value));

std::cout << "A = ";

output(A);

std::cout << "Введите радиус окружности = \t";

std::cin >> value;

if (checkInput(value)) iSpointInCircle(A, std::stod(value));

A.setX(0);

A.setY(0);

std::cout << "Введите последовательность точек, признак конца последовательности точка(0,0)\n";

Point temp;

Point point\_min;

input(temp);

double min\_distance = temp.getDistance(A);

point\_min = temp;

while ( temp.getX()!=0 || temp.getY()!=0)

{

if (min\_distance >= temp.getDistance(A))

{

min\_distance = temp.getDistance(A);

point\_min = temp;

}

input(temp);

}

if (min\_distance == 0) std::cout<<"количество точек равно 0\n";

else

{

std::cout << "Точка ближайщая к началу координат: ";

output(point\_min);

}

//тестирование класса Triangle

std::cout << "точка А:\n";

input(A);

std::cout << "точка B:\n";

input(B);

std::cout << "точка C:\n";

input(C);

Triangle ABC(A, B, C); //вызов конструктора с параметрами

std::cout << "координаты треугольника ABC:\n";

output(ABC);

Triangle ABC\_1; // вызов конструктра без параметров

std::cout << "координаты треугольника ABC\_1:\n";

output(ABC\_1);

std::cout << "координаты треугольника ABC\_1:\n";

ABC\_1.setA(ABC.getB());

output(ABC\_1);

Triangle ABC\_2(ABC); //вызов конструктора копирования

std::cout << "координаты треугольника ABC\_2:\n";

output(ABC\_2);

ABC.isTriangle() ? std::cout << "ABC треугольник\n" : std::cout << "ABC не треугольник\n";

std::cout << "Растояние, на которое будет перемещен треугольник = \t";

std::cin >> value;

if (checkInput(value)) ABC.move(std::stod(value));

output(ABC);

std::cout << "Периметр ABC = "<< ABC.getPerimeter()<<"\n";

ABC.isEqual(ABC\_2) ? std::cout << "Теугольники АВС и АВС\_1 равны\n" : std::cout<< "Теугольники АВС и АВС\_2 не равны\n";

isEqualSquare(ABC, ABC\_2);

}

catch (std::runtime\_error& ex)

{

std::cerr << ex.what()<<"\n";

exit(EXIT\_FAILURE);

}

catch (std::invalid\_argument& ex)

{

std::cerr << ex.what() << "\n";

exit(EXIT\_FAILURE);

}

return 0;

}

**Point.h**

#pragma once

#include<iostream>

#include<cmath>

#include<regex>

const std::string ERROR\_OF\_INPUT = "Некорректный ввод данных";

class Point

{

public:

Point();

Point(double x, double y);

Point(const Point& a);

~Point();

void setX(double x);

void setY(double y);

double getX()const;

double getY() const;

bool isEqual(const Point a)const;

double getDistance(const Point a)const;

void move(double k);

private:

double x\_,

y\_;

};

double checkInput(std::string &value);

void iSpointInCircle(const Point a, const double r);

void input(Point &a);

void output(const Point a);

***Point.cpp***

#include "Point.h"

Point::Point() { x\_ = 0.0; y\_ = 0.0; }

Point::Point(double x, double y) { x\_ = x; y\_ = y; }

Point::Point(const Point& a)

{

x\_ = a.x\_;

y\_ = a.y\_;

}

Point::~Point() { }

void Point::setX(double x) { x\_ = x; }

void Point::setY(double y) { y\_ = y; }

double Point::getX() const { return x\_; }

double Point::getY() const { return y\_; }

bool Point::isEqual(const Point a)const

{

return (x\_ == a.x\_ && y\_ == a.y\_) ? EXIT\_FAILURE : EXIT\_SUCCESS;

}

double Point::getDistance(const Point a) const

{

return sqrt(pow(x\_ - a.x\_, 2) + pow(y\_ - a.y\_, 2));

}

void Point::move(double k)

{

x\_ += k;

y\_ += k;

}

void iSpointInCircle(const Point a,const double r)

{

sqrt(pow(a.getX(), 2) + pow(a.getY(), 2)) <= r ? std::cout << "Точка попадает в круг радиуса " << r << "\n" :

std::cout << "Точка не попадает в круг радиуса " << r << "\n";

}

double checkInput(std::string &value)

{

std::regex regular("-?\\d+,?\\d\*");

std::regex replace("\\.");

value = std::regex\_replace(value, replace, ",");

if (std::regex\_match(value, regular)) return std::stod(value);

else throw std::invalid\_argument(ERROR\_OF\_INPUT);

}

void input(Point &a)

{

std::string value = "";

std::cout << "Введите координаты точки\n";

std::cout << " x =\t";

std::cin >>value;

a.setX(checkInput(value));

std::cout << " y =\t";

std::cin >> value;

checkInput(value);

a.setY(checkInput(value));

}

void output(const Point a)

{

std::cout << "(" << a.getX() << ", " << a.getY() << ")\n";

}

***Triangle.h***

#pragma once

#include"Point.h"

const std::string ERROR\_OF\_FIGURE = "Фигура не является треугольником";

class Triangle

{

public:

Triangle();

Triangle(Point a, Point b, Point c);

Triangle(const Triangle &a);

~Triangle();

void setA(Point a);

void setB(Point b);

void setC(Point c);

Point getA() const;

Point getB() const;

Point getC() const;

bool isTriangle()const;

void move(double k);

double getPerimeter() const;

bool isEqual(const Triangle a)const;

private:

Point a\_, b\_, c\_;

};

void isEqualSquare(const Triangle a, const Triangle b);

void output(const Triangle a);

***Triangle.cpp***

#include "Triangle.h"

Triangle::Triangle()

{

a\_.setX(0.0); a\_.setY(0.0);

b\_.setX(0.0); b\_.setY(0.0);

c\_.setX(0.0); c\_.setY(0.0);

}

Triangle::Triangle(Point a, Point b, Point c)

{

a\_ = a;

b\_ = b;

c\_ = c;

}

Triangle::Triangle(const Triangle& a)

{

a\_ = a.getA();

b\_ = a.getB();

c\_ = a.getC();

}

Triangle::~Triangle() {};

void Triangle::setA(Point a){a\_ = a;}

void Triangle::setB(Point b) {b\_ = b;}

void Triangle::setC(Point c) {c\_ = c;}

Point Triangle::getA() const {return a\_;}

Point Triangle::getB() const { return b\_;}

Point Triangle::getC() const { return c\_; }

bool Triangle::isTriangle() const

{

double AB = a\_.getDistance(b\_);

double AC = a\_.getDistance(c\_);

double BC = b\_.getDistance(c\_);

return ((AB + BC > AC) && (AB + AC > BC) && (BC + AC > AB)) ? EXIT\_FAILURE : EXIT\_SUCCESS;

}

void Triangle::move(double k)

{

a\_.move(k);

b\_.move(k);

c\_.move(k);

}

double Triangle::getPerimeter() const

{

if (isTriangle()) return a\_.getDistance(b\_) + a\_.getDistance(c\_) + b\_.getDistance(c\_);

else throw std::runtime\_error(ERROR\_OF\_FIGURE);

}

bool Triangle::isEqual(const Triangle a) const

{

return (a\_.isEqual(a.getA()) && b\_.isEqual(a.getB()) && c\_.isEqual(a.getC())) ? EXIT\_FAILURE : EXIT\_FAILURE;

}

void isEqualSquare(const Triangle a,const Triangle b)

{

if (a.isTriangle() && b.isTriangle())

{

double AB = a.getA().getDistance(a.getB());

double AC = a.getA().getDistance(a.getC());

double BC = a.getB().getDistance(a.getC());

double square\_1 = sqrt(0.5 \* a.getPerimeter() \* (0.5 \* a.getPerimeter() - AB) \* (0.5 \* a.getPerimeter() - AC) \* (0.5 \* a.getPerimeter() - BC));

AB = b.getA().getDistance(b.getB());

AC = b.getA().getDistance(b.getC());

BC = b.getB().getDistance(b.getC());

double square\_2 = sqrt(0.5 \* b.getPerimeter() \* (0.5 \* b.getPerimeter() - AB) \* (0.5 \* b.getPerimeter() - AC) \* (0.5 \* b.getPerimeter() - BC));

square\_1 == square\_2 ? std::cout << "Площади равны\n" : std::cout << "Площади не равны\n";

}

else throw std::runtime\_error(ERROR\_OF\_FIGURE);

}

void output(const Triangle a)

{

output(a.getA());

output(a.getB());

output(a.getC());

}

**Приложение A**

***Все переменные, вводимые с консоли (значения Х и Y точек, расстояние, на которое будет перемещена точка или треугольник, радиус окружности), должны быть заданы дробными числам:***

*1. Некорректный ввод данных*







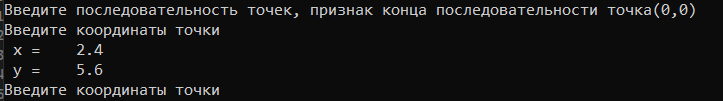


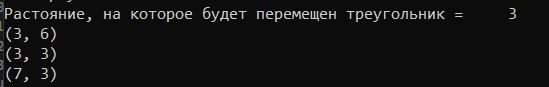


*2. Данные заданы корректно*

****

****

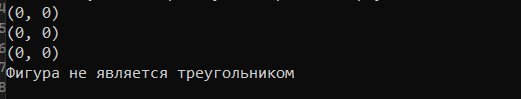
****

****

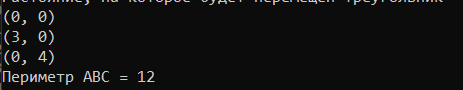
**Приложение B**

***Для нахождения периметра заданные точки должны образовывать треугольник***

*1.* *Заданные точки не образуют треугольник*



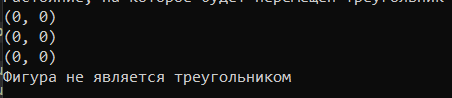
*2. Заданные точки образуют треугольник*

**

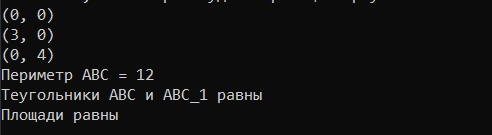
**Приложение C**

***Для сравнения площадей обе заданные фигуры должны являться треугольниками***

1. Точки хотя бы одной из фигур не образуют треугольник



2. Точки обеих фигур образуют треугольники



**Вывод**

В ходе работы:

1. Были изучены классы, были написаны 2 класса Point ( переменные типа Point хранят координаты точек) и Triangle ( переменные типа Triangle задают треугольник)
2. Были изучены и реализованы 3 типа конструкторов: конструктор без параметров, конструктор с параметрами и конструктор копирования
3. Были изучены и реализованы деструкторы
4. Были изучены модификаторы доступа (public, private, protected)
5. Были реализованы закрытые свойства классов
6. Были реализованы открытые методы этих классов
7. Была использована конструкция try … catch, которая позволяет эффективнее обрабатывать возникающие в ходе работы программы ошибки