ФИО: Ребдев Павел Александрович
Группа: 5130904/30008
Лабораторная работа: «Перегрузка операций»

Постановка задачи

Разработать детальные требования и тест план для следующей задачи:

Point. Реализовать перегрузку операторов «==», «<», «>=», «+» и «+=» для класса Point, провести тестирование операторов

Детальные требования

1. Оператор «==»:
   1.1. Если соответствующие координаты двух объектов класса Point равны, то возвратить 1
   1.2. Если хотя бы одна из координат не равна соответствующей координате второго объекта, то возвратить 0
2. Оператор «<»:
   2.1. Если сумма квадратов координат первого объекта меньше суммы квадратов координат второго объекта, то возвратить 1
   2.2. Если сумма квадратов координат первого объекта не меньше суммы квадратов координат второго объекта, то возвратить 0
3. Оператор «>=»:
   3.1. Если сумма квадратов координат первого объекта не меньше суммы квадратов координат второго объекта, то возвратить 1
   3.2. Если сумма квадратов координат первого объекта меньше суммы квадратов координат второго объекта, то возвратить 0
4. Оператор «+»:
   4.1. Если k - число, то возвратить объект класса Point с координатами (x + k) и (y + k)

5. Оператор «+=»:
   5.1. Если k - число, то прибавить к координатам текущей точки k и вернуть ссылку на текущей обьект

Тест-план

Проверка детальных требований с помощью тест-плана:

#	**Описание**	**Результат**
1.1	Если соответствующие координаты двух объектов класса Point равны, то возвратить 1	Point point1(2.0, 1.0) Point point2(2.0, 1.0) **point1 == point2** **Expected:** `1`
1.2	Если хотя бы одна из координат не равна соответствующей координате второго объекта, то возвратить 0	Point point1(2.0, 1.0) Point point2(1.0, 2.0) **point1 == point2** **Expected:** `0`
2.1	Если сумма квадратов координат первого объекта меньше суммы квадратов координат второго объекта, то возвратить 1	Point point1(2.0, 1.0) Point point2(2.0, 2.0) **point1 < point2** **Expected:** `1`
2.2	Если сумма квадратов координат первого объекта не меньше суммы квадратов координат второго объекта, то возвратить 0	Point point1(2.0, 2.0) Point point2(2.0, 1.0) **point1 < point2** **Expected:** `0`
3.1	Если сумма квадратов координат первого объекта не меньше суммы квадратов координат второго объекта, то возвратить 1	Point point1(4.0, 2.0) Point point2(2.0, 4.0) **point1 >= point2** **Expected:** `1`
3.2	Если сумма квадратов координат первого	Point point1(2.0, 1.0)

	объекта меньше суммы квадратов координат второго объекта, то возвратить 0	Point point2(2.0, 2.0) **point1 >= point2** **Expected:** `0`
4.1	Если k - число, то возвратить объект класса Point с координатами (x + k) и (y + k)	k = 12.3 Point point1(1.0, -47.92) **point1 + k** **Expected:** `point(13.3, -35.62)`
5.1	Если k - число, то прибавить к координатам текущей точки k и вернуть ссылку на текущей обьект	k = -19.73 Point point1(0.0, 128.77) **point1 += k** **Expected:** `point1(-19.73, -109.04)`

Исходные тексты программы

Файлы с исходными текстами лабораторной работы (полагаем <ROOT> для папки в котором располагаются исходные тексты):

./<ROOT>/main.cpp

```
#include <iostream>
#include <cstddef>
#include <ctime>
#include <limits>
#include "Point.h"
#include "pointFunction.hpp"

int main()
{
  Point * pointArr = nullptr;
  Point * newArr = nullptr;
  size_t numberOfPoints = 0;
  do
  {
    newArr = new Point[numberOfPoints + 1];
    for (size_t i = 0; i < numberOfPoints; ++i)
    {
      newArr[i].setX(pointArr[i].getX());
      newArr[i].setY(pointArr[i].getY());
    }

    try
    {
      input(newArr[numberOfPoints]);
    }
    catch (const std::logic_error & e)
    {
      delete[] pointArr;
      delete[] newArr;

      std::cerr << e.what();
      return 1;
    }

    delete[] pointArr;
    pointArr = newArr;
    newArr = nullptr;
```

```
    numberOfPoints += 1;
     }  while  ((pointArr[numberOfPoints  -  1].getX()  !=  0.0)  ||
(pointArr[numberOfPoints - 1].getY() != 0.0));

  srand(time(NULL));
    size_t  numberOfFirstPoint  =  (rand()  %  (numberOfPoints  -  1)),
numberOfSecondPoint = (rand() % (numberOfPoints - 1));
  while (numberOfSecondPoint == numberOfFirstPoint)
  {
    numberOfSecondPoint = (rand() % (numberOfPoints - 1));
  }
   std::cout  <<  "Number  of  points:  "  <<  numberOfFirstPoint  <<  "  "  <<
numberOfSecondPoint << '\n';

  std::cout << "x and y cordinats: ";
  output(pointArr[numberOfFirstPoint]);
  std::cout << " ";
  output(pointArr[numberOfSecondPoint]);
  std::cout << '\n';

  std::cout << "first point == second point: ";
            std::cout      <<      (pointArr[numberOfFirstPoint]      ==
pointArr[numberOfSecondPoint]) << '\n';

  std::cout << "first point < second point: ";
             std::cout      <<      (pointArr[numberOfFirstPoint]      <
pointArr[numberOfSecondPoint]) << '\n';

  std::cout << "first point >= second point: ";
            std::cout      <<      (pointArr[numberOfFirstPoint]      >=
pointArr[numberOfSecondPoint]) << '\n';

  double k = (rand() % 100) + double(rand() % 100)/100;
  std::cout << "Points + k(" << k << ") ";
  output(pointArr[numberOfFirstPoint] + k);
  std::cout << " ";
  pointArr[numberOfSecondPoint] += k;
  output(pointArr[numberOfSecondPoint]);
  std::cout << '\n';

  return 0;
}
```

./<ROOT>/Point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
class Point
{
  public:
    Point();
    Point(double x, double y);
    Point(Point & point);
```

```
    ~Point();

    bool operator==(const Point & point) const;
    bool operator<(const Point & point) const;
    bool operator>=(const Point & point) const;
    Point operator+(double k);
    Point & operator+=(double k);

    void setX(double x);
    void setY(double y);
    double getX() const;
    double getY() const;

    bool isEqual(Point point);
    double getDistance(Point point);
    void move(double distance);
  private:
    double x_, y_;
};

#endif
```

./<ROOT>/Point.cpp

```
#include "Point.h"
#include <cmath>

Point::Point():
 x_(0.0),
 y_(0.0)
 {};
Point::Point(double x, double y):
 x_(x),
 y_(y)
 {};
Point::Point(Point & point):
 x_(point.getX()),
 y_(point.getY())
 {};
Point::~Point()
{
 x_ = 0.0;
};

bool Point::operator==(const Point & point) const
{
 return ((x_ == point.getX()) && (y_ == point.getY()));
};
bool Point::operator<(const Point & point) const
{
 return ((x_ * x_ + y_ * y_) < (point.getX() * point.getX() + point.getY() * point.getY()));
};
bool Point::operator>=(const Point & point) const
{
 return (point < *this);
};
Point Point::operator+(double k)
{
```

```
  Point point((x_ + k), (y_ + k));
  return point;
};
Point & Point::operator+=(double k)
{
  x_ += k;
  y_ += k;
  return *this;
};

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(Point point)
{
  return (x_ == point.getX()) && (y_ == point.getY());
};
double Point::getDistance(Point point)
{
   return  std::sqrt((x_  -  point.getX())  *  (x_  -  point.getX())  +  (y_  -  point.getY())  *  (y_  -
point.getY()));
};
void Point::move(double distance)
{
  x_ += distance;
  y_ += distance;
};
```

./<ROOT>/pointFunction.hpp

```
#ifndef POINTFUNCTION_HPP
#define POINTFUNCTION_HPP
#include <cstddef>
#include "Point.h"

bool isPointInCircle(const Point & point, size_t radius);
void input (Point & point);
void output (const Point & point);
#endif
```

./<ROOT>/pointFunction.cpp

```
#include "pointFunction.hpp"
#include <iostream>
#include <cmath>
```

```
bool isPointInCircle(const Point & point, size_t radius)
{
  double x = point.getX(), y = point.getY();
  return std::sqrt(x * x + y * y) <= radius;
};

void input (Point & point)
{
  double x = 0.0, y = 0.0;
  std::cin >> x >> y;
  if (!std::cin)
  {
    throw std::logic_error("Bad input!");
  }
  point.setX(x);
  point.setY(y);
};

void output (const Point & point)
{
  std::cout << point.getX() << " ";
  std::cout << point.getY() << " ";
};
```