Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа
по курсу «ООП»

Тема:
Перегрузка операторов.

Студент:

Кузьмичев А. Н.

Группа:

М80-206Б-18

Преподаватель:

Журавлев А.А.

Вариант:

12

Оценка:

Дата:

Москва
2019

1. Код программы на языке C++:

Rectangle.h:

```
#ifndef LABA1_HOME_RECTANGLE_H
#define LABA1_HOME_RECTANGLE_H
#include <iostream>

class Rectangle {
private:
    double left_lower_x;
    double left_lower_y;
    double right_upper_x;
    double right_upper_y;

public:
    Rectangle(double llx, double lly, double rrx, double rry) :
left_lower_x(llx), left_lower_y(lly),

right_upper_x(rrx), right_upper_y(rry) {}

    Rectangle() = default;

    void Input(std::istream&); // ввод с клавиатуры
    void Output(std::ostream&) const; // вывод на экран
    double Square() const; // подсчет площади
    double Perimeter() const; // подсчет периметра
    Rectangle Move(double dx, double dy) const; // перемещение
    Rectangle Resize(double alpha) const; // изменение размеров
    bool Compare_by_square(const Rectangle& other) const; //
сравнение по площади
    bool Compare_by_perimeter(const Rectangle& other) const; //
сравнение по периметру
    Rectangle Cross(const Rectangle& other) const; // Пересечение
двух прямоугольников
    Rectangle Min(const Rectangle& other) const; // минимальный
прямоугольник, включающий оба прямугольника

    operator double () const;
    Rectangle operator++();
    Rectangle operator++(int);
    bool operator>(const Rectangle& other) const;
    bool operator<(const Rectangle& other) const;
    bool operator==(const Rectangle& other) const;
    bool operator!=(const Rectangle& other) const;
    Rectangle operator&(const Rectangle& other) const;
    Rectangle operator|(const Rectangle& other) const;


   friend std::ostream& operator << (std::ostream& stream, const
Rectangle& out);
```

```
    friend std::istream& operator >> (std::istream& stream,
Rectangle& in);
};
Rectangle operator"" _square(long double n);
```

```
#endif //LABA1_HOME_RECTANGLE_H
```

Rectangle.cpp:

```
#include <iostream>
#include "Rectangle.h"
#include <algorithm>

void Rectangle::Input(std::istream& is) {
    is >> left_lower_x >> left_lower_y;
    is >> right_upper_x >> right_upper_y;
}
void Rectangle::Output(std::ostream& os) const {
    os << "A - " << "(" << left_lower_x << "," << left_lower_y <<
")" << std::endl;
    os << "B - " << "(" << left_lower_x << "," << right_upper_y <<
")" << std::endl;
    os << "C - " << "(" << right_upper_x << "," << right_upper_y <<
")" << std::endl;
    os << "D - " << "(" << right_upper_x << "," << left_lower_y <<
")" << std::endl;
}

double Rectangle::Square() const {
    double a = right_upper_y - left_lower_y;
    double b = right_upper_x - left_lower_x;
    double square = a * b;
    return square;
}

double Rectangle::Perimeter() const {
    double a = right_upper_y - left_lower_y;
    double b = right_upper_x - left_lower_x;
    double perimeter = 2 * (a + b);
    return perimeter;
}
```

```
Rectangle Rectangle::Move(double dx, double dy) const {
    Rectangle result;
    result.left_lower_x = left_lower_x  + dx;
    result.left_lower_y = left_lower_y + dy;
    result.right_upper_x = right_upper_x + dx;
    result.right_upper_y = right_upper_y + dy;
    return result;
}

Rectangle Rectangle::Resize(double alpha) const {
    Rectangle result;
    result.left_lower_x = left_lower_x * alpha;
    result.left_lower_y = left_lower_y * alpha;
    result.right_upper_x = right_upper_x * alpha;
    result.right_upper_y = right_upper_y * alpha;
    return result;
}
bool Rectangle::Compare_by_square(const Rectangle& other) const {
    return Square() > other.Square();
}


bool Rectangle::Compare_by_perimeter(const Rectangle& other) const {
    return Perimeter() > other.Perimeter();
}

Rectangle Rectangle::Min(const Rectangle& other) const {
    Rectangle result;
    result.left_lower_x = std::min(left_lower_x,
other.left_lower_x);
    result.left_lower_y = std::min(left_lower_y,
other.left_lower_y);
    result.right_upper_x = std::max(right_upper_x,
other.right_upper_x);
    result.right_upper_y = std::max(right_upper_y,
other.right_upper_y);
    return result;

}

Rectangle Rectangle::Cross(const Rectangle& other) const {
    Rectangle result;
    result.left_lower_x = std::max(left_lower_x,
other.left_lower_x);
    result.left_lower_y = std::max(left_lower_y,
other.left_lower_y);
    result.right_upper_x = std::min(right_upper_x,
other.right_upper_x);
    result.right_upper_y = std::min(right_upper_y,
other.right_upper_y);
```

```
    if (result.left_lower_x > result.right_upper_x ||
result.left_lower_y > result.right_upper_y) {
        result.right_upper_x = result.left_lower_x;
        result.right_upper_y = result.left_lower_y;
    }
    return result;
}



Rectangle::operator double() const {
    return Square();
}

Rectangle Rectangle::operator++(){
    right_upper_x++;
    right_upper_y++;
    return *this;
}


//Rectangle Rectangle::operator++(int) {
  //  Rectangle result = *this;
    //right_upper_x++;
    //right_upper_y++;
    //return result;
//}

Rectangle Rectangle::operator++(int) {
  Rectangle temp = *this;
  ++(*this);
  return temp;
}

bool Rectangle::operator>(const Rectangle& other) const{
    return Square() > other.Square();
}

bool Rectangle::operator<(const Rectangle& other) const{
    return Square() < other.Square();
}

bool Rectangle::operator==(const Rectangle& other) const {
    return Square() == other.Square();
}

bool Rectangle::operator!=(const Rectangle& other) const {
    return Square() != other.Square();
}

std::istream& operator >> (std::istream& stream, Rectangle& in)
```

```
{
    stream >> in.left_lower_x >> in.left_lower_y >> in.right_upper_x
>> in.right_upper_y;
    return stream;
}

std::ostream& operator << (std::ostream& stream, const Rectangle&
out) {
    out.Output(stream);
    return stream;
}

Rectangle Rectangle::operator&(const Rectangle& other) const{
    return Cross(other);
}

Rectangle Rectangle::operator|(const Rectangle& other) const{
    return Min(other);
}

Rectangle operator"" _square(long double n) {
    Rectangle lit(0, 0, n, n);
    return lit;
}
```

Main.cpp:

```
#include <iostream>
#include "Rectangle.h"

int main() {
    Rectangle rectangle1;
    Rectangle rectangle2;

    std::cin >> rectangle1;
    std::cin >> rectangle2;
    std::cout << rectangle1 << std::endl;
    std::cout << rectangle2 << std::endl;


    std::cout << (rectangle1 & rectangle2) << "\n";
    std::cout << (rectangle1 | rectangle2) << "\n";
    std::cout << (rectangle1 < rectangle2) << "\n";
    std::cout << (rectangle1 == rectangle2) << "\n";
    std::cout << double(rectangle1) << "\n";
    std::cout << rectangle1++ << "\n";
```

```
    std::cout << ++rectangle1 << "\n";

std::cout << 1._square << "\n";



}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.12)
project(laba_oop_2)

set(CMAKE_CXX_STANDARD 14)

add_executable(laba_oop_2 main.cpp Rectangle.cpp Rectangle.h)
```

2. Ссылка на репозиторий на GitHub.

https://github.com/poisoned-monkey/oop_exercise_02.git

3. Набор testcases.

Test 1
0 0 2 2
1 1 3 3

Test 2

0 0 3 3
4 4 6 6

4. Результаты выполнения тестов.

Test 1

A - (0,0)
B - (0,2)
C - (2,2)
D - (2,0)

A - (1,1)
B - (1,3)
C - (3,3)
D - (3,1)

A - (1,1)
B - (1,2)
C - (2,2)
D - (2,1)

A - (0,0)
B - (0,3)
C - (3,3)
D - (3,0)

0
1
4
A - (0,0)
B - (0,2)
C - (2,2)
D - (2,0)

A - (0,0)
B - (0,4)
C - (4,4)
D - (4,0)

A - (0,0)
B - (0,1)
C - (1,1)
D - (1,0)

Test 2

A - (0,0)
B - (0,3)
C - (3,3)
D - (3,0)

A - (4,4)
B - (4,6)

C - (6,6)
D - (6,4)

A - (4,4)
B - (4,4)
C - (4,4)
D - (4,4)

A - (0,0)
B - (0,6)
C - (6,6)
D - (6,0)

0
0
9
A - (0,0)
B - (0,3)
C - (3,3)
D - (3,0)

A - (0,0)
B - (0,5)
C - (5,5)
D - (5,0)

A - (0,0)
B - (0,1)
C - (1,1)
D - (1,0)

Вывод:

Выполняя данную лабораторную работу, я познакомился с механизмом перегрузки операторов в C++. Я понял, что перегрузкой нужно пользоваться только при крайней необходимости, чтобы не усложнять код. Также я научился работать с пользовательскими литералами.