Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 2: «Операторы, литералы»

Группа:	М8О-108Б-18, №12
Студент:	Коростелев Дмитрий Васильевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	14.10.2019

Москва, 2019

1. Задание

1. Разработать класс Rectangle, представляющий собой прямоугольник со сторонами, параллельными осям координат. Поля – координаты левого нижнего и правого верхнего угла. Требуется реализовать следующие методы: вычисление площади и периметра, перемещения вдоль осей, изменение размеров, сравнение по площади и по периметру. Реализовать метод получения прямоугольника, представляющего общую часть (пересечение) двух прямоугольников. Реализовать метод объединения двух прямоугольников: наименьший прямоугольник, включающего оба заданных прямоугольника.

2. Необходимо реализовать:

3. - операцию приведения к типу double, вычисляющую площадь прямоугольника;

4. - операцию префиксного и постфиксного инкремента, увеличивающую одновременно размеры сторон прямоугольника.

5. - операции сравнения (больше, меньше, равно);

Необходимо реализовать пользовательский литерал для работы с константами типа Rectangle.

2. Адрес репозитория на GitHub

https://github.com/Dmitry4K/oop_exercise_2

3. Код программы на C++

Main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "rectangle.h"
#include <locale>

int main(int argc, char *argv[]) {
	int a, b, c, d;
	std::cout << "Enter Coordinate of First Rectangle: ";
	std::cin >> a >> b >> c >> d;
	Rectangle rec1 = Rectangle(a, b, c, d);
	std::cout << "Enter Coordinate of Second Rectangle: ";
	std::cin >> a >> b >> c >> d;
	Rectangle rec2 = Rectangle(a, b, c, d);
	std::cout << "First Rectangle: " << rec1 << std::endl;
	std::cout << "Second Rectangle: " << rec2 << std::endl;
	Rectangle res = Rectangle();
	res = rec1 | rec2;
	std::cout << "Composition of Rectangles: " << res << std::endl;
	res = rec1 & rec2;
	std::cout << "Intersection of Rectangles: " << res << std::endl;
	std::cout << "Increment of First Rectagngle: " << ++rec1 << std::endl;
```

```
	std::cout << "Comparison: " << (rec1 == rec2) << std::endl;
	std::cout << "Square of First Rectangle: " << (double)rec1 << std::endl;
	Rectangle rec3 = "0 1 2 3"_rtg;
	int rec3S = "0 1 2 3"_rtgSquare;
	std::cout << "Third Rectangle: " << rec3 << " Square: "<< rec3S<<std::endl;
	Rectangle rec4 = "0 1 27 0"_rtg;
	int rec4S = "0 1 27 0"_rtgSquare;
	std::cout << "Fourth Rectangle: " << rec4 << " Square: " << rec4S << std::endl;

	Rectangle rec5 = "-110 -2"_rtg;
	int rec5S = "-110 -2"_rtgSquare;
	std::cout << "Fifth Rectangle: " << rec5 << " Square: " << rec5S << std::endl;
	system("pause");
	return 0;
}
```

Rectangle.h

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

class Coordinate {
private:
	int x;
	int y;
public:
	Coordinate();
	Coordinate(int x, int y);
	int getX() const;
	int getY() const;
	void setX(int x);
	void setY(int y);
	void set(Coordinate f);
	Coordinate& operator+=(const Coordinate& b);
	Coordinate& operator+=(const int& b);
	Coordinate& operator-=(const int& b);
	Coordinate& operator-=(const Coordinate& b);
	Coordinate& operator++();
	Coordinate& operator--();
	Coordinate& operator--(int);
	Coordinate& operator++(int);
	friend std::ostream& operator<< (std::ostream &out, const Coordinate &point);
	friend std::istream& operator>> (std::istream &in, Coordinate &point);
};

Coordinate operator+(const Coordinate& a, const Coordinate& b);
Coordinate operator+(const Coordinate& a, int& b);
Coordinate operator+(int& a, const Coordinate& b);
Coordinate operator-(const Coordinate& a, int& b);
Coordinate operator-(int& a, const Coordinate& b);

class Rectangle {
protected:
	Coordinate first;
	Coordinate second;
public:
	Rectangle();
	Rectangle(Coordinate f, Coordinate s);
	Rectangle(int fX, int fY, int sX, int sY);
	Coordinate getFirstCoordinate() const;
	Coordinate getSecondCoordinate() const;
	int square() const;
	int perimetr() const;
```

```
        Rectangle intersection(const Rectangle rec) const;
        Rectangle composition(const Rectangle rec) const;
        void shift(int onX, int onY);
        void changeRectangle(Coordinate f, Coordinate s);
        bool compareSquare(Rectangle rec);
        bool comparePerimetr(Rectangle rec);

        Rectangle& operator++();
        Rectangle& operator++(int);
        Rectangle& operator+=(const Rectangle& b);
        Rectangle& operator-=(const Rectangle& b);
        Rectangle& operator+=(const int& b);
        Rectangle& operator-=(const int& b);
        Rectangle& operator--();
        Rectangle& operator--(int);
        friend bool operator>(const Rectangle &a, const Rectangle& b);
        friend bool operator== (const Rectangle &a, const Rectangle& b);
        friend bool operator<(const Rectangle &a, const Rectangle& b);
        friend bool operator<=(const Rectangle &a,const Rectangle& b);
        friend bool operator>=(const Rectangle &a,const Rectangle& b);

        friend std::ostream& operator<< (std::ostream &out, const Rectangle &point);
        friend std::istream& operator>> (std::istream &in, Rectangle &point);
        operator double ();
};

Rectangle operator-(const Rectangle &a, const Rectangle& b);
Rectangle operator+(const Rectangle& a, int& b);
Rectangle operator+(int& a, const Rectangle& b);
Rectangle operator-(const Rectangle& a, int& b);
Rectangle operator-(int& a, const Rectangle& b);
Rectangle operator+(const Rectangle &a, const Rectangle& b);

Rectangle operator&(const Rectangle &a, const Rectangle& b);
Rectangle operator| (const Rectangle &a, const Rectangle& b);
Rectangle operator "" _rtg(const char* str,size_t size);
int operator "" _rtgSquare(const char* str, size_t size);
#endif
```

Rectangle.cpp

```
#include "rectangle.h"
#include <sstream>
int max(int a, int b) {
    return a > b ? a : b;
}

int min(int a, int b) {
    return a > b ? b : a;
}

int abs(int a) {
    return a > 0 ? a : -a;
}


Coordinate::Coordinate() {
    this->x = 0;
    this->y = 0;
}

Coordinate::Coordinate(int x, int y) {
    this->x = x;
    this->y = y;
```

```
}

int Coordinate::getX() const {
    return x;
}
int Coordinate::getY() const {
    return y;
}
void Coordinate::setX(int x) {
    this->x = x;
}
void Coordinate::setY(int y) {
    this->y = y;
}
void Coordinate::set(Coordinate f) {
    this->x = f.x;
    this->y = f.y;
}

Coordinate& Coordinate::operator+=(const Coordinate& b) {
    x += b.x;
    y += b.y;
    return *this;
}

Coordinate& Coordinate::operator++() {
    ++x;
    ++y;
    return *this;
}
Coordinate& Coordinate::operator++(int) {
    Coordinate result = Coordinate(this->x++, this->y++);
    *this = result;
    return *this;
}

Coordinate& Coordinate::operator+=(const int& b) {
    x += b;
    y += b;
    return *this;
}

Coordinate operator+(const Coordinate& a, const Coordinate& b) {
    Coordinate result = Coordinate(a);
    result += b;
    return result;
}

Coordinate operator+(const Coordinate &a, int &b) {
    Coordinate result = Coordinate(a);
    result += b;
    result += b;
    return result;
}

Coordinate operator+(int& a, const Coordinate& b) {
    Coordinate result = Coordinate(b);
    result += a;
    result += a;
    return result;
}

Coordinate& Coordinate::operator-=(const int& b) {
    x -= b;
    y -= b;
```

```
    return *this;
}

Coordinate& Coordinate::operator-=(const Coordinate& b) {
    x -= b.x;
    y -= b.y;
    return *this;
}

Coordinate operator-(const Coordinate& a, int& b) {
    Coordinate result = Coordinate(a);
    result -= b;
    result -= b;
    return result;
}
Coordinate operator-(int& a, const Coordinate& b) {
    Coordinate result = Coordinate(b);
    result -= a;
    result -= a;
    return result;
}
Coordinate& Coordinate::operator--() {
    --x;
    --y;
    return *this;
}
Coordinate& Coordinate::operator--(int) {
    Coordinate result = Coordinate(this->x--,this->y--);
    *this = result;
    return *this;
}

std::ostream& operator<< (std::ostream &out, const Coordinate &point) {
    out << "X: " << point.x << " Y: " << point.y;
    return out;
}


std::istream& operator>> (std::istream &in, Coordinate &point) {
    in >> point.x >> point.y;
    return in;
}


Rectangle::Rectangle() {
    this->first = Coordinate();
    this->second = Coordinate();
}
Rectangle::Rectangle(Coordinate f, Coordinate s) {
    this->first = Coordinate();
    this->second = Coordinate();
    first.set(f);
    second.set(s);
}
Rectangle::Rectangle(int fX, int fY, int sX, int sY) {
    this->first = Coordinate(fX, fY);
    this->second = Coordinate(sX, sY);
}

Coordinate Rectangle::getFirstCoordinate() const {
    return this->first;
}
Coordinate Rectangle::getSecondCoordinate() const {
    return this->second;
}
```

```
int Rectangle::square() const{
    return abs((first.getX() - second.getX())*(first.getY() - second.getY()));
}
int Rectangle::perimetr() const {
    return 2 * (abs(first.getX() - second.getX()) + abs(first.getY() - second.getY()));
}

Rectangle Rectangle::composition(const Rectangle rec) const {
    int left = min(first.getX(), rec.first.getX());
    int right = max(second.getX(), rec.second.getX());
    int top = max(second.getY(), rec.second.getY());
    int bottom = min(first.getX(), rec.first.getY());

    Coordinate f;
    f.setX(left);
    f.setY(bottom);
    Coordinate s;
    s.setX(right);
    s.setY(top);
    Rectangle result{ f,s };
    return result;

}
Rectangle Rectangle::intersection(const Rectangle rec) const {
    Rectangle result{};
    int left = max(this->first.getX(), rec.first.getX());
    int top = min(this->second.getY(), rec.second.getY());
    int right = min(this->second.getX(), rec.second.getX());
    int bottom = max(this->first.getY(), rec.first.getY());

    int width = right - left;
    int height = top - bottom;

    if (!((width < 0) || (height < 0))) {
        Coordinate f;
        f.setX(left);
        f.setY(bottom);
        Coordinate s;
        s.setX(right);
        s.setY(top);
        result.changeRectangle(f, s);
    }

    return result;
}

void Rectangle::shift(int onX, int onY) {
    first.setX(first.getX() + onX);
    second.setY(first.getY() + onY);
}
void Rectangle::changeRectangle(Coordinate f, Coordinate s) {
    first.set(f);
    second.set(s);
}
bool Rectangle::compareSquare(Rectangle rec) {
    return this->square() == rec.square();
}
bool Rectangle::comparePerimetr(Rectangle rec) {
    return this->perimetr() == rec.perimetr();
}

Rectangle operator+(const Rectangle& a, int& b) {
    Rectangle result = Rectangle(a);
    result += b;
```

```
    return result;
}
Rectangle operator+(int& b,const  Rectangle& a) {
    Rectangle result = Rectangle(a);
    result += b;
    return result;
}

Rectangle& Rectangle::operator++() {
    ++first;
    ++second;
    return *this;
}
Rectangle& Rectangle::operator++(int) {
    Rectangle result = Rectangle(this->first++, this->second++);
    *this = result;
    return *this;
}

Rectangle operator-(const Rectangle& a, int& b) {
    Rectangle result = Rectangle(a);
    result += b;
    return result;
}
Rectangle operator-(int& b, const Rectangle& a) {
    Rectangle result = Rectangle(a);
    result += b;
    return result;
}

Rectangle& Rectangle::operator--() {
    --first;
    --second;
    return *this;
}
Rectangle& Rectangle::operator--(int) {
    Rectangle result = Rectangle(this->first--, this->second--);
    *this = result;
    return *this;
}

Rectangle operator+(const Rectangle &a, const Rectangle& b) {
    Rectangle result = Rectangle(a);
    result += b;
    return result;
}
Rectangle& Rectangle::operator+=(const Rectangle& b) {
    first += b.first;
    second += b.second;
    return *this;
}

Rectangle& Rectangle::operator+=(const int& b) {
    first += b;
    second += b;
    return *this;
}
Rectangle& Rectangle::operator-=(const Rectangle& b) {
    first -= b.first;
    second -= b.second;
    return *this;
}
Rectangle& Rectangle::operator-=(const int& b) {
    first -= b;
    second -= b;
```

```
    return *this;
}
Rectangle operator-(const Rectangle &a, const Rectangle& b) {
    Rectangle result = Rectangle(a);
    result -= b;
    return result;
}

bool operator>(const Rectangle &a,const Rectangle& b) {
    return a.square() > b.square() ? true : false;
}
bool operator== (const Rectangle &a,const  Rectangle& b) {
    return a.square() == b.square() ? true : false;
}
bool operator<(const Rectangle &a,const Rectangle& b) {
    return a.square() < b.square() ? true : false;
}
bool operator<=(const Rectangle &a,const Rectangle& b) {
    return a.square() <= b.square() ? true : false;
}
bool operator>=(const Rectangle &a,const  Rectangle& b) {
    return a.square() >= b.square() ? true : false;
}

std::ostream& operator<< (std::ostream &out, const Rectangle &point) {
    out << "First coordinate: " << point.first << " Second coordinate: " << point.second;
    return out;
}

std::istream& operator>> (std::istream &in, Rectangle &point) {
    in >> point.first >> point.second;
    return in;
}

Rectangle::operator double () {
    int result = this->square();
    return result;
}

/*Rectangle operator "" _rtg(const char* str, size_t size) {
    bool reg = false;
    int * a = new int[4];
    for (int i = 0; i < 4; i++)
        a[i] = 0;
    int count = 0, number = 0;
    for (int i = 0; i < (int)size; i++) {
        if (str[i] == ':') {
                if ((count + 1) == 4)
                        break;
                if (reg) number = -number;
                a[count] = number;
                count++;
                number = 0;
                reg = false;
        }
        else {
                if (str[i] == '-')
                        reg = true;
                else {
                        number *= 10;
                        number += str[i] - '0';
                }
        }
    }
    if (reg) number = -number;
```

```
    a[count] = number;
       return Rectangle(a[0],a[1],a[2],a[3]);

}*/

Rectangle operator&(const Rectangle &a,const  Rectangle& b) {
    return Rectangle(a.intersection(b));
}
Rectangle operator| (const Rectangle &a,const Rectangle& b) {
    return Rectangle(a.composition(b));
}

Rectangle operator "" _rtg(const char* str, size_t size) {
    std::istringstream is(str);
    Rectangle result = Rectangle();
    is >> result;
    return result;
}


int operator "" _rtgSquare(const char* str, size_t size) {
    std::istringstream is(str);
    Rectangle result = Rectangle();
    is >> result;
    return result.square();
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(lab2)
set(SOURCE_EXE main_io.cpp)
set(SOURCE_LIB rectangle.cpp)

add_library(rectangle STATIC ${SOURCE_LIB})

add_executable(main ${SOURCE_EXE})

target_link_libraries(main rectangle)
```

4. Результаты выполнения тестов

Тест 1 :

Enter Coordinate of First Rectangle: 0 0 1 2
Enter Coordinate of Second Rectangle: 0 1 1 1
First Rectangle: First coordinate: X: 0 Y: 0 Second coordinate: X: 1 Y: 2
Second Rectangle: First coordinate: X: 0 Y: 1 Second coordinate: X: 1 Y: 1
Composition of Rectangles: First coordinate: X: 0 Y: 0 Second coordinate: X: 1 Y: 2
Intersection of Rectangles: First coordinate: X: 0 Y: 1 Second coordinate: X: 1 Y: 1
Increment of First Rectagngle: First coordinate: X: 1 Y: 1 Second coordinate: X: 2 Y: 3

Comparison: 0
Square of First Rectangle: 2
Third Rectangle: First coordinate: X: 0 Y: 1 Second coordinate: X: 2 Y: 3 Square: 4
Fourth Rectangle: First coordinate: X: 0 Y: 1 Second coordinate: X: 27 Y: 0
Square: 27
Fifth Rectangle: First coordinate: X: -110 Y: -2 Second coordinate: X: 0 Y: 0
Square: 220
Тест 2:

Enter Coordinate of First Rectangle: 0 0 2 2
Enter Coordinate of Second Rectangle: 0 0 1 4
First Rectangle: First coordinate: X: 0 Y: 0 Second coordinate: X: 2 Y: 2
Second Rectangle: First coordinate: X: 0 Y: 0 Second coordinate: X: 1 Y: 4
Composition of Rectangles: First coordinate: X: 0 Y: 0 Second coordinate: X: 2 Y: 4
Intersection of Rectangles: First coordinate: X: 0 Y: 0 Second coordinate: X: 1 Y: 2
Increment of First Rectagngle: First coordinate: X: 1 Y: 1 Second coordinate: X: 3 Y: 3
Comparison: 1
Square of First Rectangle: 4
Third Rectangle: First coordinate: X: 0 Y: 1 Second coordinate: X: 2 Y: 3 Square: 4
Fourth Rectangle: First coordinate: X: 0 Y: 1 Second coordinate: X: 27 Y: 0 Square: 27
Fifth Rectangle: First coordinate: X: -110 Y: -2 Second coordinate: X: 0 Y: 0 Square: 220

5. Объяснение результатов работы программы

Программа запрашивает на вход координаты прямоугольника из стандартного потока вывода, по 4 на каждый прямоугольник, далее, происходит вывод всей полученной информации прямоугольника, инкремент прямоугольника, пересечение и объединение прямоугольников, вычисление площади. Далее с помощью литерального значения создается экземпляр класса с заданными в литеральной строке данными, также описан литерал, который по указанным координатам находит площадь прямоугольника. Вывод осуществляется в стандартный поток вывод с помощью объекта std::cout и перегруженными операторами логического сдвига/

Арифметические и логические операторы классов Coordinate и Rectangle.

Оператор	Функция оператора
operator+, operator-,	Сложение полей Coordinate Сложение полей Rectangle

operator++, operator--,	Постфиксный и префиксный инкремент полей Coordinate и Rectangle
Operator &, operator|	У Coordinate сложение полей, у Rectangle вычисление объединения и пересечения.
Operator (double)	Определен у rectangle, вычисляет площадь прямоугольника.
Операторы сравнения	Определены у rectangle, сравнивает по площади прямоугольников.

6. Вывод

Благодаря перегрузке логических операторов программист, который будет использовать данный класс, получает более гибкое и простое управление над экземплярами класса, а благодаря литералам пользователь в одной строке может задать параметры экземпляра и начать работу с ним.