МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторные работы №1-3

По дисциплине «Программирование»

Выполнил студент группы №M3106

Казаков Никита Андреевич

Проверил

Повышев Владислав Вячеславович

Санкт-Петербург
2020

Лабораторная работа №3
Перегрузка операторов.

Согласно варианту описать указанные типы данных и поместить их в отдельный заголовочный файл, в нем же описать операторы, указанные в варианте. Реализацию функций поместить с отдельный cpp файл.
Написать программу, проверяющую правильность работы – для наглядности и лучшего усвоения материала использовать как явный, так и не явный метод вызова функций операторов (см. пример в конце задания).

main.cpp

```
#include <iostream>
#include <vector>
#include "functions.h"
using namespace std;
int main() {
    vector <int> new_pos = {1, 2};
    Triangle mem(9, 15, 20, 13, 20, 5);
    Triangle mem_2(10, 16, 21, 14, 21, 6);
    Array_int array_1(5, 1);
    Array_int array_2(6, 2);
    if (mem_2 == mem)
        cout << mem.area() << '\n';
    if (mem < mem_2)
        cout << mem << '\n';
    mem.transport(new_pos);
    cout << mem << '\n';
    if (array_1 < array_2)
        cout << array_1 << '\n';
    cout << array_2 << '\n';
    array_1 = array_1 + array_2;
    cout << array_1 << '\n';
    return 0;
}
```

functions.cpp

```
#include <iostream>
#include <cmath>
#include <vector>
#include "functions.h"
Triangle::Triangle (double x_1, double x_2, double y_1, double y_2, double z_1,
double z_2) : a_x(x_1), a_y(x_2),

b_x(y_1), b_y(y_2), c_x(z_1), c_y(z_2){}
double Triangle::area () const {
    double a_side = sqrt(pow(this->a_x - this->b_x, 2) + pow(this->a_y - this-
>b_y, 2));
    double b_side = sqrt(pow(this->b_x - this->c_x, 2) + pow(this->b_y - this-
>c_y, 2));
```

```
    double c_side = sqrt(pow(this->c_x - this->a_x, 2) + pow(this->c_y - this-
>a_y, 2));
    double per = (a_side + b_side + c_side) / 2.;
    return sqrt(per * (per - a_side) * (per - b_side) * (per - c_side));
}
void Triangle::transport (std::vector <int> new_pos) {
    if (!new_pos.empty()) {
        this->a_x += new_pos[0];
    }
    if (new_pos.size() > 1) {
        this->a_y += new_pos[1];
    }
    if (new_pos.size() > 2) {
        this->b_x += new_pos[2];
    }
    if (new_pos.size() > 3) {
        this->b_y += new_pos[3];
    }
    if (new_pos.size() > 4) {
        this->c_x += new_pos[4];
    }
    if (new_pos.size() > 5) {
        this->c_y += new_pos[5];
    }
}
Array_int &Array_int::operator= (const Array_int &right) {
    if (this != &right) {
        this->size = right.size;
        delete[] this->body;
        this->body = new int[this->size];
        for (int i = 0; i < this->size; i++) {
            this->body[i] = right.body[i];
        }
    }
    return *this;
}
```

functions.h

```
#pragma once
class Triangle {
public:
    Triangle (double, double, double, double, double, double);
    double area () const;
    void transport (std::vector <int>);
    friend std::ostream &operator<< (std::ostream &out, const Triangle &value) {
        out << '[' << value.a_x << ", " << value.a_y << ']' << '[' << value.b_x
<< ", " << value.b_y << ']'
        << '[' << value.c_x << ", " << value.c_y << ']';
        return out;
    }
    friend bool operator> (const Triangle &left, const Triangle &right) {
        return left.area() > right.area();
    }
    friend bool operator< (const Triangle &left, const Triangle &right) {
```

```
        return left.area() < right.area();
    }
    friend bool operator== (const Triangle &left, const Triangle &right) {
        return left.area() == right.area();
    }
    friend bool operator!= (const Triangle &left, const Triangle &right) {
        return left.area() != right.area();
    }
private:
    double a_x;
    double a_y;
    double b_x;
    double b_y;
    double c_x;
    double c_y;
};
class Array_int {
public:
    Array_int (int new_size, int temp_value) {
        if (new_size > 100) {
            std::cerr << "Array is too big: " << new_size << "/100" << '\n';
            std::exit(EXIT_FAILURE);
        }
        this->size = new_size;
        this->body = new int [new_size];
        for (int i = 0; i < this->size; i++) {
            this->body[i] = temp_value;
        }
    }
    ~Array_int() {
        delete [] this->body;
    }
    friend std::ostream &operator<< (std::ostream &out, const Array_int &value)
{
        out << '{';
        for (int i = 0; i < value.size; i++) {
            out << value.body[i];
            if (i != value.size - 1) {
                out << ", ";
            }
        }
        out << '}';
        return out;
    }
    friend Array_int operator+ (const Array_int &left, const Array_int &right) {
        Array_int temp(left.size + right.size, left.body[0]);
        for (int i = 0; i < left.size; i++) {
            temp.body[i] = left.body[i];
        }
        for (int i = left.size; i < left.size + right.size; i++) {
            temp.body[i] = right.body[i - left.size];
        }
        return temp;
    }
    Array_int &operator= (const Array_int &);
    friend bool operator> (const Array_int &left, const Array_int &right) {
```

```
        return left.size > right.size;
    }
    friend bool operator< (const Array_int &left, const Array_int &right) {
        return left.size < right.size;
    }
    friend bool operator== (const Array_int &left, const Array_int &right) {
        return left.size == right.size;
    }
    friend bool operator!= (const Array_int &left, const Array_int &right) {
        return left.size != right.size;
    }
private:
    int size;
    int *body;
};
```