Main.cpp

```cpp
#include "header.h"
#include "array.h"

using namespace std;

void first_task(){
    Matrix m0(); // ZERO constructor
    Matrix m1(name_enter()); // NORMAL constructor
    Matrix m2(name_enter());
    Matrix m3 = m1 * m2; // matrix * matrix
    Matrix m4 = m1 + m2; // matrix + matrix
    Matrix m5 = m1 - m2; // matrix - matrix
    m1 *= 5; // matrix * num

}

void second_task() {
    std::cout << std::endl;
    std::cout << "    Task 2:" << std::endl;
    Array First, Second;
    std::cin >> First >> Second;
    if (First < Second)
        std::cout << "First array is smaler than second" << std::endl;
    if (First > Second)
        std::cout << "First array is larger than second" << std::endl;
    if (First == Second)
        std::cout << "Arrays are same" << std::endl;
    if (First != Second)
        std::cout << "Arrays are different" << std::endl;
    std::cout << std::endl;
    std::cout << First << std::endl;
    std::cout << std::endl;
    std::cout << Second << std::endl;
    std::cout << std::endl;
    Array Third = First.operator+(Second);
    std::cout << Third << std::endl;
}

int main() {

    first_task();
    second_task();

    return 0;
}
```

First_task:

Header.h

```
#ifndef LAB3_HEADER_H
#define LAB3_HEADER_H
#include <iostream>
#include <string>

using namespace std;
string name_enter();

class Matrix {
private:
    string name;
    int matrix[3][3];
public:
    Matrix(); // ZERO constructor
    Matrix(const string &add_name); // NORMAL constructor
    Matrix operator+(const Matrix &matrix_right); // matrix + matrix
    Matrix operator-(const Matrix &matrix_right); // matrix - matrix
    Matrix operator*=(const int &num); // matrix * num
    Matrix operator*(const Matrix &matrix_right); // matrix * matrix
    bool operator==(const Matrix &matrix_right);
    bool operator!=(const Matrix &matrix_right);
    bool operator>(Matrix &matrix_right);
    bool operator<(Matrix &matrix_right);
    int deter(int (&matrix)[3][3]);
    void show_matrix();
};

#endif
```

S_procedures.cpp

```
#include <iostream>
#include <string>
#include "header.h"

string name_enter() {
    cout << "Enter matrix name:" << endl;
    string name;
    cin >> name;
    return name;
}
```

C_procedures.cpp

```cpp
#include <iostream>
#include <string>
#include "header.h"

using namespace std;

Matrix::Matrix() { // ZERO constructor
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            this->matrix[i][j] = 0;
        }
    }
}
Matrix::Matrix(const string &add_name) { // NORMAL constructor

    this->name = add_name;
    cout << "Matrix elems:" << endl;
    int num;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cin >> num;
            this->matrix[i][j] = num;
        }
    }
    show_matrix();
}
Matrix Matrix::operator+(const Matrix &matrix_right) { // matrix + matrix
    Matrix new_matrix;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            new_matrix.matrix[i][j] = this->matrix[i][j] + matrix_right.matrix[i][j];
        }
    }

    new_matrix.show_matrix();
    return new_matrix;
}
Matrix Matrix::operator-(const Matrix &matrix_right) { // matrix - matrix
    Matrix new_matrix;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            new_matrix.matrix[i][j] = this->matrix[i][j] - matrix_right.matrix[i][j];
        }
    }

    new_matrix.show_matrix();
    return new_matrix;
}
Matrix Matrix::operator*=(const int &num) { // matrix * num

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            this->matrix[i][j] *= num;
        }
    }
    show_matrix();
    return *this;
}
Matrix Matrix::operator*(const Matrix &matrix_right) { // matrix * matrix

    Matrix new_matrix;
    new_matrix.name = name_enter();

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            new_matrix.matrix[i][j] = 0;
            for (int k = 0; k < 3; k++) {
                new_matrix.matrix[i][j] += matrix[i][k] * matrix_right.matrix[k][j];
            }
        }
    }

    new_matrix.show_matrix();
    return new_matrix;
}
bool Matrix::operator==(const Matrix &matrix_right) {

    bool check = true;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (this->matrix[i][j] != matrix_right.matrix[i][j]) {
                check = false;
                break;
            }
        }
    }

    return check;
}
bool Matrix::operator!=(const Matrix &matrix_right) {

    bool check = false;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (this->matrix[i][j] != matrix_right.matrix[i][j]) {
                check = true;
                break;
            }
        }
    }

    return check;
}
bool Matrix::operator>(Matrix &matrix_right) {
    int d1 = deter(this->matrix);
    int d2 = deter(matrix_right.matrix);

    if (d1 > d2) {
        return true;
    } else {
        return false;
    }
}
bool Matrix::operator<(Matrix &matrix_right) {
    int d1 = deter(this->matrix);
    int d2 = deter(matrix_right.matrix);
    if (d1 > d2) {
        return false;
    } else {
        return true;
    }
}
int Matrix::deter(int (&add_matrix)[3][3]) {
    int det = add_matrix[0][0] * add_matrix[1][1] * add_matrix[2][2] -
add_matrix[0][0] * add_matrix[1][2] * add_matrix[2][1]
                - add_matrix[0][1] * add_matrix[1][0] * add_matrix[2][2] +
add_matrix[0][1] * add_matrix[1][2] * add_matrix[2][0]
                + add_matrix[0][2] * add_matrix[1][0] * add_matrix[2][1] -
add_matrix[0][2] * add_matrix[1][1] * add_matrix[2][0];
    return det;
}
void Matrix::show_matrix() {
    cout << "---------------------------------------" << endl;
    cout << "MATRIX NAME: " << this->name << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << this->matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << "---------------------------------------" << endl;
}
```

Second_task:

Array.h

```
#pragma once
#include <fstream>

class Array
{
private:
    int* array;
    size_t size;
public:
    Array();
    explicit Array(size_t size);
    Array(Array const&);
    ~Array();
    size_t Size() const;
    void push_back(int v);
    void resize(int newSize);
    int& operator[](int i);
    Array operator+(const Array&);
    friend std::istream& operator>>(std::istream&, Array&);
};

std::ostream& operator<<(std::ostream&, Array&);
bool operator==(Array&, Array&);
bool operator!=(Array&, Array&);
bool operator>(Array&, Array&);
bool operator<(Array&, Array&);
```

Array.cpp

```
#include "array.h"

Array::Array(){
    size = 0;
}

Array::Array(size_t size){
    this->size = size;
    array = new int[size];
    for (size_t i = 0; i < size; i++) {
        array[i] = 0;
    }
}
Array::Array(Array const& a){
    size = a.Size();
    array = new int[size];
    for (size_t i = 0; i < size; i++) {
        array[i] = a.array[i];
    }
}

Array::~Array(){
    delete[] array;
}

size_t Array::Size() const {
    return size;
}
void Array::push_back(int v) {
    int* temp = new int[size + 1];
    for (size_t i = 0; i < size; i++) {
        temp[i] = array[i];
    }
    temp[size] = v;
    delete[] array;
    array = temp;
    size++;
}
void Array::resize(int newSize){
    int* temp = new int[newSize];
    size_t minSize = newSize > size ? size : newSize;
    for (size_t i = 0; i < minSize; i++) {
        temp[i] = array[i];
    }
    delete[] array;
    array = temp;
    size = newSize;
}
int& Array::operator[] (int i) {
    if (i < size && i >= 0)
        return array[i];
}
std::istream& operator>>(std::istream& stream, Array& a){
    size_t size;
    stream >> size;
    for (size_t i = 0; i < size; i++){
        int temp;
        stream >> temp;
        a.push_back(temp);
    }
    return stream;
}
std::ostream& operator<<(std::ostream& stream, Array& a){
    stream << "Length: " << a.Size() << "\n" << "[ ";
    for (size_t i = 0; i < a.Size(); i++){
        stream << a[i] << " ";
    }
    stream << "]";
    return stream;
}
Array Array::operator+(const Array& b){
    int tmp = this->size;
    this->resize(this->size + b.size);
    for (int i = 0; i < b.size; i++) {
        this->array[tmp + i] = b.array[i];
    }
    return *this;
}

bool operator==(Array& a, Array& b){
    return a.Size() == b.Size();
}

bool operator!=(Array& a, Array& b){
    return a.Size() != b.Size();
}

bool operator<(Array& a, Array& b){
    return a.Size() < b.Size();
}

bool operator>(Array& a, Array& b){
    return a.Size() > b.Size();
}
```

Ввод	Вывод
Название и содержимое матриц	Название и содержимое матриц
Название итоговой матрицы	Результат перемножения матриц, умножения на вещ. Число, вычитание и сложение матриц, сравнение
Размер и содержимое двух целочисленных масивов	Сравнение длин массивов их сложение

ЛР№3

Микаилов Микаил Аскерович
группа №M3110

Проверил преподаватель:
Повышев Владислав Вячеславович