**Отчет о выполнении лабораторной работы 2**

**Олейников Артём, группа НБИ-БД-23-02**

Код:

#include <iostream>

#include <cstdlib>

#include <ctime>

class Vect {

private:

int dim;

double\* data;

public:

Vect(int d) : dim(d), data(new double[d]) {}

Vect(const Vect& other) : dim(other.dim), data(new double[other.dim]) {

for (int i = 0; i < dim; ++i) {

data[i] = other.data[i];

}

}

~Vect() {

delete[] data;

}

int getDim() const {

return dim;

}

double\* getData() const {

return data;

}

void display() const {

for (int i = 0; i < dim; ++i) {

std::cout << data[i] << " ";

}

std::cout << std::endl;

}

Vect& operator=(const Vect& other) {

if (this == &other) return \*this;

if (dim != other.dim) {

delete[] data;

dim = other.dim;

data = new double[dim];

}

for (int i = 0; i < dim; ++i) {

data[i] = other.data[i];

}

return \*this;

}

Vect operator-() const {

Vect result(dim);

for (int i = 0; i < dim; ++i) {

result.data[i] = -data[i];

}

return result;

}

Vect operator+(const Vect& v) const {

Vect result(dim);

for (int i = 0; i < dim; ++i) {

result.data[i] = data[i] + v.data[i];

}

return result;

}

Vect operator-(const Vect& v) const {

Vect result(dim);

for (int i = 0; i < dim; ++i) {

result.data[i] = data[i] - v.data[i];

}

return result;

}

double operator\*(const Vect& v) const {

double result = 0;

for (int i = 0; i < dim; ++i) {

result += data[i] \* v.data[i];

}

return result;

}

friend Vect operator\*(double k, const Vect& v);

};

Vect operator\*(double k, const Vect& v) {

Vect result(v.getDim());

for (int i = 0; i < v.getDim(); ++i) {

result.getData()[i] = k \* v.getData()[i];

}

return result;

}

Vect operator\*(const Vect& v, double k) {

return k \* v;

}

class Matr {

private:

int dim;

double\*\* a;

public:

Matr(int d) : dim(d), a(new double\*[d]) {

for (int i = 0; i < d; ++i) {

a[i] = new double[d];

}

}

Matr(const Matr& other) : dim(other.dim), a(new double\*[other.dim]) {

for (int i = 0; i < dim; ++i) {

a[i] = new double[dim];

for (int j = 0; j < dim; ++j) {

a[i][j] = other.a[i][j];

}

}

}

~Matr() {

for (int i = 0; i < dim; ++i) {

delete[] a[i];

}

delete[] a;

}

int getDim() const {

return dim;

}

double\*\* getData() const {

return a;

}

void display() const {

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

std::cout << a[i][j] << " ";

}

std::cout << std::endl;

}

}

Matr& operator=(const Matr& other) {

if (this == &other) return \*this;

if (dim != other.dim) {

for (int i = 0; i < dim; ++i) {

delete[] a[i];

}

delete[] a;

dim = other.dim;

a = new double\*[dim];

for (int i = 0; i < dim; ++i) {

a[i] = new double[dim];

}

}

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

a[i][j] = other.a[i][j];

}

}

return \*this;

}

Matr operator-() const {

Matr result(dim);

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

result.a[i][j] = -a[i][j];

}

}

return result;

}

Matr operator+(const Matr& m) const {

Matr result(dim);

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

result.a[i][j] = a[i][j] + m.a[i][j];

}

}

return result;

}

Matr operator-(const Matr& m) const {

Matr result(dim);

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

result.a[i][j] = a[i][j] - m.a[i][j];

}

}

return result;

}

friend Matr operator\*(double k, const Matr& m);

friend Matr operator\*(const Matr& m, double k);

friend Matr operator\*(const Matr& m1, const Matr& m2);

friend Vect operator\*(const Matr& m, const Vect& v);

};

Matr operator\*(double k, const Matr& m) {

Matr result(m.dim);

for (int i = 0; i < m.dim; ++i) {

for (int j = 0; j < m.dim; ++j) {

result.a[i][j] = k \* m.a[i][j];

}

}

return result;

}

Matr operator\*(const Matr& m, double k) {

return k \* m;

}

Matr operator\*(const Matr& m1, const Matr& m2) {

if (m1.dim != m2.dim) {

std::cerr << "Ошибка: Несовпадение размерностей для умножения матриц\n";

exit(1);

}

Matr result(m1.dim);

for (int i = 0; i < m1.dim; ++i) {

for (int j = 0; j < m1.dim; ++j) {

result.a[i][j] = 0;

for (int k = 0; k < m1.dim; ++k) {

result.a[i][j] += m1.a[i][k] \* m2.a[k][j];

}

}

}

return result;

}

Vect operator\*(const Matr& m, const Vect& v) {

if (m.getDim() != v.getDim()) {

std::cerr << "Ошибка: Несовпадение размерностей для умножения матрицы на вектор\n";

exit(1);

}

Vect result(v.getDim());

for (int i = 0; i < m.getDim(); ++i) {

result.getData()[i] = 0;

for (int j = 0; j < m.getDim(); ++j) {

result.getData()[i] += m.getData()[i][j] \* v.getData()[j];

}

}

return result;

}

// Функция для генерации случайных чисел в диапазоне от min до max (включительно)

int getRandom(int min, int max) {

return min + rand() % (max - min + 1);

}

int main() {

srand(time(nullptr)); // Инициализация генератора случайных чисел

int dim;

std::cout << "Введите размерность векторов и матриц: ";

std::cin >> dim;

// Создание и заполнение первого вектора случайными не нулевыми элементами

Vect v1(dim);

for (int i = 0; i < dim; ++i) {

v1.getData()[i] = getRandom(0, 100);

}

// Создание и заполнение второго вектора случайными элементами

Vect v2(dim);

for (int i = 0; i < dim; ++i) {

v2.getData()[i] = getRandom(0, 100);

}

// Создание и заполнение первой матрицы случайными элементами

Matr m1(dim);

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

m1.getData()[i][j] = getRandom(0, 100);

}

}

// Создание и заполнение второй матрицы случайными элементами

Matr m2(dim);

for (int i = 0; i < dim; ++i) {

for (int j = 0; j < dim; ++j) {

m2.getData()[i][j] = getRandom(0, 100);

}

}

// Вывод первого вектора и первой матрицы

std::cout << "Сгенерированный первый вектор:" << std::endl;

v1.display();

std::cout << "Сгенерированный второй вектор:" << std::endl;

v2.display();

double scalar;

std::cout << "Введите скаляр: ";

std::cin >> scalar;

Vect v\_scalar\_mul = v1 \* scalar;

std::cout << "Умножение вектора на скаляр: ";

v\_scalar\_mul.display();

double dot\_product = v1 \* v2;

std::cout << "Скалярное произведение векторов: " << dot\_product << std::endl;

Vect v3 = -v1;

Vect v4(dim);

v4 = v1;

std::cout << "Унарный минус вектора: ";

v3.display();

std::cout << "Присваивание первого вектора: ";

v4.display();

std::cout << "Сгенерированная первая матрица:" << std::endl;

m1.display();

std::cout << "Сгенерированная вторая матрица:" << std::endl;

m2.display();

Matr m\_diff = m1 - m2;

Matr m\_scalar\_mul = m1 \* scalar;

Matr m\_product = m1 \* m2;

Matr m3 = -m1;

Matr m4(dim);

m4 = m1;

std::cout << "Разность матриц: ";

m\_diff.display();

std::cout << "Умножение матрицы на скаляр: ";

m\_scalar\_mul.display();

std::cout << "Произведение матриц: ";

m\_product.display();

std::cout << "Унарный минус матрицы: ";

m3.display();

std::cout << "Присваивание первой матрицы второй: ";

m4.display();

Vect v5 = m1 \* v1;

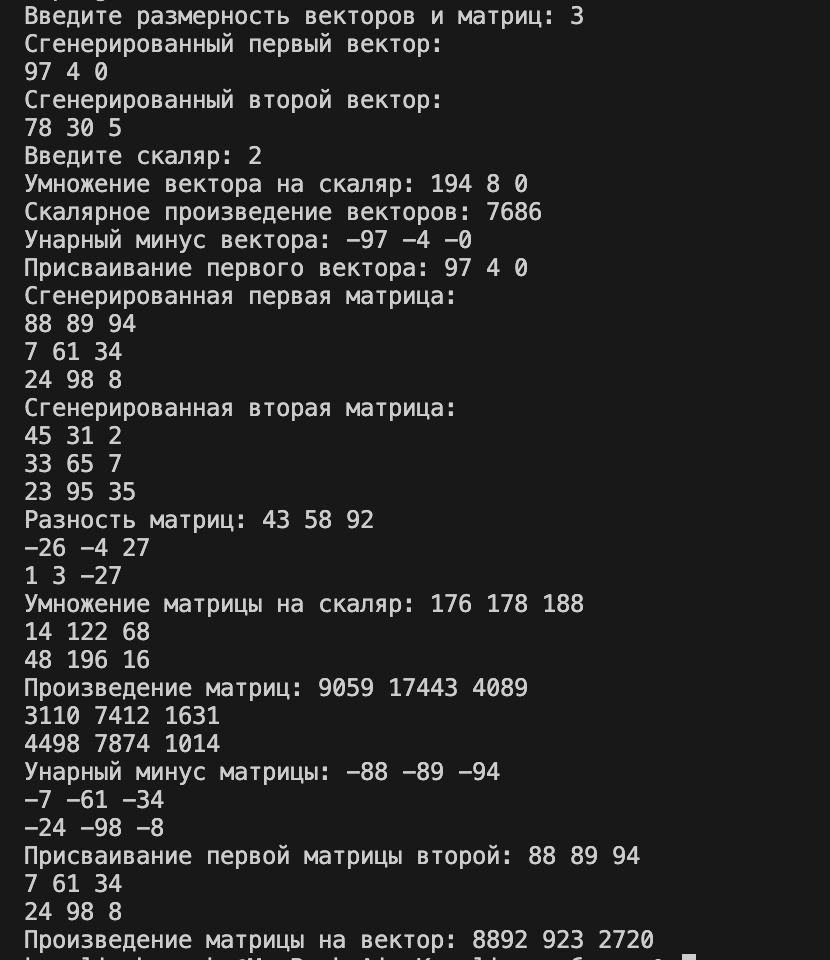
std::cout << "Произведение матрицы на вектор: ";

v5.display();

return 0;

}

Вывод:



**Отчет о выполнении лабораторной работы**

**Тема:** Создание классов Vect и Matr для работы с векторами и матрицами.

**Описание алгоритма:**

1. **Класс Vect:**
   * Поля класса:
     + int dim: хранит размерность вектора.
     + double\* data: указатель на массив, хранящий компоненты вектора.
     + int num: переменная, которая хранит порядковый номер созданного объекта класса.
     + static int count: статическое поле, содержащее количество созданных объектов класса.
   * Методы класса:
     + Конструктор Vect(int d): выделяет память под массив data размерности d.
     + Конструктор копирования Vect(const Vect& other): создает копию вектора other.
     + Деструктор ~Vect(): освобождает память, выделенную для массива data.
     + int getDim() const: возвращает размерность вектора.
     + double\* getData() const: возвращает указатель на массив компонент вектора.
     + void display() const: выводит компоненты вектора на экран.
     + Vect& operator=(const Vect& other): перегрузка оператора присваивания.
     + Vect operator-() const: перегрузка унарного минуса.
     + Vect operator+(const Vect& v) const: перегрузка оператора сложения векторов.
     + Vect operator-(const Vect& v) const: перегрузка оператора вычитания векторов.
     + double operator\*(const Vect& v) const: перегрузка оператора скалярного произведения векторов.
     + friend Vect operator\*(double k, const Vect& v): перегрузка оператора умножения вектора на скаляр (дружественная функция).
     + friend Vect operator\*(const Vect& v, double k): перегрузка оператора умножения вектора на скаляр (дружественная функция).
2. **Класс Matr:**
   * Поля класса:
     + int dim: хранит размерность квадратной матрицы.
     + double\*\* a: указатель на двумерный массив, хранящий элементы матрицы.
     + int num: переменная, которая хранит порядковый номер созданного объекта класса.
     + static int count: статическое поле, содержащее количество созданных объектов класса.
   * Методы класса:
     + Конструктор Matr(int d): выделяет память под двумерный массив a размерности d×d.
     + Конструктор копирования Matr(const Matr& other): создает копию матрицы other.
     + Деструктор ~Matr(): освобождает память, выделенную для двумерного массива a.
     + int getDim() const: возвращает размерность матрицы.
     + double\*\* getData() const: возвращает указатель на двумерный массив элементов матрицы.
     + void display() const: выводит элементы матрицы на экран.
     + Matr& operator=(const Matr& other): перегрузка оператора присваивания.
     + Matr operator-() const: перегрузка унарного минуса.
     + Matr operator+(const Matr& m) const: перегрузка оператора сложения матриц.
     + Matr operator-(const Matr& m) const: перегрузка оператора вычитания матриц.
     + friend Matr operator\*(double k, const Matr& m): перегрузка оператора умножения матрицы на скаляр (дружественная функция).
     + friend Matr operator\*(const Matr& m, double k): перегрузка оператора умножения матрицы на скаляр (дружественная функция).
     + friend Matr operator\*(const Matr& m1, const Matr& m2): перегрузка оператора умножения матриц (дружественная функция).
     + friend Vect operator\*(const Matr& m, const Vect& v): перегрузка оператора умножения матрицы на вектор (дружественная функция).

**Структура программы:**

1. Включение необходимых заголовочных файлов <iostream>, <cstdlib> и <ctime>.
2. Определение класса Vect с полями dim, data, num и count, а также методами для работы с векторами.
3. Определение класса Matr с полями dim, a, num и count, а также методами для работы с матрицами.
4. Реализация основных операций с векторами и матрицами: сложение, вычитание, умножение векторов и матриц, умножение матрицы на вектор и на скаляр.
5. Функция getRandom(int min, int max), генерирующая случайные числа в заданном диапазоне.
6. Основная функция main, в которой создаются объекты классов Vect и Matr, выполняются операции над ними и выводятся результаты на экран.