МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**«Структуры хранения матриц специального вида»**

**Выполнил:** студент группы 3822Б1ПР2

Коровин Никита Олегович

Нижний Новгород  
2023

**Содержание**

[1. Введение 3](#_Toc150690389)

[2. Постановка задачи 4](#_Toc150690390)

[3. Руководство пользователя 5](#_Toc150690391)

[4. Руководство программиста 6](#_Toc150690392)

[4.1. Описание структуры программы 6](#_Toc150690393)

[4.2. Описание структур данных 7](#_Toc150690394)

[4.3. Описание алгоритмов 10](#_Toc150690395)

[5. Эксперименты 11](#_Toc150690396)

[6. Заключение 12](#_Toc150690397)

[7. Литература 13](#_Toc150690398)

[8. Приложение 14](#_Toc150690399)

# Введение

Матрица в математике — это прямоугольная таблица, которая состоит из элементов, расположенных в строках и столбцах. Эти элементы могут быть числами или математическими выражениями. Матрицы играют важную роль в различных областях математики, включая линейную алгебру, системы линейных уравнений и векторные пространства.

Действия с матрицами должны выполняться согласно правилам матричной алгебры. Если матрица имеет одинаковое количество строк и столбцов, то есть , она называется квадратной матрицей. В случае, когда количество строк не совпадает с количеством столбцов, матрица является прямоугольной.

Ключевым понятием является главная диагональ матрицы, которая включает элементы . Когда все элементы матрицы вне главной диагонали равны нулю, такая матрица определяется как диагональная.

Также существует понятие верхнетреугольной матрицы. Это матрица, в которой все элементы ниже главной диагонали равны нулю. Это означает, что любой элемент матрицы, для которого , будет равен нулю. Эти определения и свойства матриц являются фундаментальными и широко используются в математическом анализе и приложениях.

Можно также понимать матрицу как совокупность векторов. Вектор в математике представляет собой набор элементов, таких как числа или математические выражения, упорядоченных в определенной последовательности. Если рассмотреть матрицу, состоящую из строк и столбцов, то её можно описать как ряд векторов, количество которых соответствует числу столбцов , и каждый из этих векторов содержит элементов, соответствующих строкам матрицы. То есть, каждый столбец матрицы — это вектор, состоящий из элементов каждой строки.

# Постановка задачи

*Задача:* В данной лабораторной работе ставится задача создания программных средств, поддерживающих эффективное хранение матриц и выполнение основных операций над ними:

• сложение/вычитание;

• умножение;

• копирование;

• сравнение.

Программные средства должны содержать:

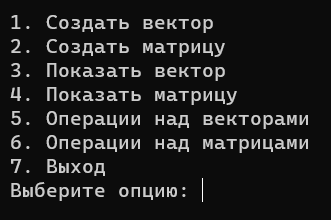
• класс Вектор (на шаблонах);

• класс Матрица (на шаблонах);

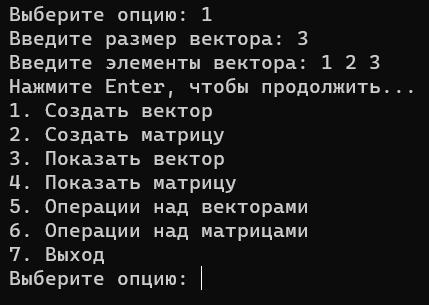
• тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними

# Руководство пользователя

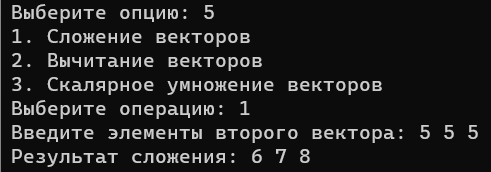
Программа предоставляет инструменты для создания и управления векторами и матрицами. Пользователи могут создавать новые векторы или матрицы, просматривать их содержимое и выполнять различные математические операции.



После запуска программы отображается меню с опциями для работы с векторами и матрицами. Для создания вектора или матрицы необходимо выбрать соответствующий пункт меню и ввести размерность, а затем элементы. Каждый элемент вводится последовательно с новой строки или разделяется пробелами в одной строке.



При выборе функций для отображения, на экране будет представлено текущее состояние выбранного вектора или матрицы. Все операции, включая сложение, вычитание, скалярное умножение и умножение матриц, выполняются после подтверждения введённых данных. Для завершения работы с программой предусмотрен пункт выхода из главного меню.



# **Руководство программиста**

## Описание структуры программы

Программа состоит из четырёх основных файлов:

1. tmatrix.h: Этот заголовочный файл содержит шаблонные классы TDynamicVector и TDynamicMatrix, которые представляют собой реализацию динамического вектора и матрицы соответственно. Классы включают методы для базовых операций, таких как индексация, сравнение, арифметические операции и методы для ввода/вывода.
2. test\_tmatrix.cpp: Файл содержит модульные тесты для класса TDynamicMatrix, используя библиотеку Google Test. Тесты покрывают различные аспекты, включая создание матриц, их копирование, доступ к элементам, арифметические операции и проверки на граничные условия.
3. test\_tvector.cpp: Аналогично, этот файл содержит модульные тесты для класса TDynamicVector. Тесты оценивают функциональность векторов, включая создание, копирование, работу с элементами, арифметические операции и соответствие размерностей при выполнении операций.
4. sample\_matrix.cpp: Главный файл исходного кода, который демонстрирует использование классов TDynamicVector и TDynamicMatrix в приложении. Он содержит интерактивное меню для создания, отображения и выполнения операций с векторами и матрицами. Пользователи могут взаимодействовать с программой через командную строку для выполнения желаемых действий.

Каждый файл теста содержит набор тест-кейсов, которые систематически проверяют корректность логики и устойчивость работы методов и операций, определённых в классах TDynamicVector и TDynamicMatrix.

Используемые в файлах тестов макросы ASSERT\_NO\_THROW, ASSERT\_ANY\_THROW, EXPECT\_EQ и EXPECT\_NE позволяют оценить, соответствует ли поведение классов ожидаемому при различных условиях использования.

Все файлы вместе формируют комплексное решение для работы с векторами и матрицами, предоставляя как инструменты для непосредственной работы с данными, так и средства для их тестирования.

## Описание структур данных

Описание шаблонных классов TDynamicVector и TDynamicMatrix, используемых для работы с динамически выделенными массивами и матрицами:

**TDynamicVector**

Шаблонный класс для представления динамического массива (вектора).

*Поля:*

size\_t sz; - хранит размер вектора.

T\* pMem; - указатель на массив элементов типа T.

*Методы:*

*Конструкторы:*

TDynamicVector(size\_t size = 1); - инициализирует вектор заданного размера. Выбрасывает исключение, если размер меньше или равен нулю или больше MAX\_VECTOR\_SIZE.

TDynamicVector(T\* arr, size\_t s); Конструктор копирования, копирующий элементы из существующего массива arr.

TDynamicVector(const TDynamicVector& v); Конструктор копирования, создающий копию существующего вектора.

TDynamicVector(TDynamicVector&& v) noexcept; Конструктор перемещения, использующий r-value ссылки.

*Деструктор:*

~TDynamicVector(); - освобождает память, занятую вектором.

*Операторы:*

TDynamicVector& operator=(const TDynamicVector& v); - оператор присваивания для копирования.

TDynamicVector& operator=(TDynamicVector&& v) noexcept; - оператор присваивания для перемещения.

T& operator[](size\_t ind); - предоставляет доступ к элементам вектора по индексу. Выбрасывает исключение при выходе за границы.

const T& operator[](size\_t ind) const; - константная версия оператора индексации.

bool operator==(const TDynamicVector& v) const noexcept; - сравнение на равенство.

bool operator!=(const TDynamicVector& v) const noexcept; - сравнение на неравенство.

*Арифметические операторы:*

TDynamicVector operator+(T val); - сложение вектора с скаляром.

TDynamicVector operator-(T val); - вычитание скаляра из вектора.

TDynamicVector operator\*(T val); - умножение вектора на скаляр.

TDynamicVector operator+(const TDynamicVector& v); - сложение двух векторов.

TDynamicVector operator-(const TDynamicVector& v); - вычитание двух векторов.

T operator\*(const TDynamicVector& v); - скалярное произведение векторов.

*Дружественные функции:*

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept; - обменивает содержимое двух векторов.

friend istream& operator>>(istream& istr, TDynamicVector& v); - ввод вектора из потока.

friend ostream& operator<<(ostream& ostr, const TDynamicVector& v); - вывод вектора в поток.

**TDynamicMatrix**

*Поля:*

Использует поля pMem и sz из базового класса TDynamicVector.

*Методы:*

*Конструктор:*

TDynamicMatrix(size\_t s = 1); - инициализирует матрицу размером s x s. Выбрасывает исключение при некорректном размере.

*Индексация:*

Использует операторы индексации от базового класса TDynamicVector.

*Сравнение:*

bool operator==(const TDynamicMatrix& m) const noexcept;

*Матрично-скалярные операции:*

TDynamicMatrix operator\*(const T& val); - умножение матрицы на скаляр.

Матрично-векторные операции:

TDynamicVector<T> operator\*(const TDynamicVector<T>& v); - умножение матрицы на вектор.

Матрично-матричные операции:

TDynamicMatrix operator+(const TDynamicMatrix& m); - сложение матриц.

TDynamicMatrix operator-(const TDynamicMatrix& m); - вычитание матриц.

TDynamicMatrix operator\*(const TDynamicMatrix& m); - умножение матриц.

Ввод/вывод:

friend istream& operator>>(istream& istr, TDynamicMatrix& m); - ввод матрицы из потока.

friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& m); - вывод матрицы в поток.

## Описание алгоритмов

*Умножение вектора на скаляр:*

Алгоритм проходит по каждому элементу вектора и умножает его на заданное скалярное значение. Результатом является новый вектор, где каждый элемент равен соответствующему элементу исходного вектора, умноженному на скаляр.

*Сложение двух векторов:*

Для сложения двух векторов одинакового размера алгоритм последовательно складывает соответствующие пары элементов обоих векторов. Если размеры векторов не совпадают, операция не может быть выполнена и генерируется исключение.

*Скалярное произведение векторов:*

Скалярное произведение двух векторов одинаковой размерности включает умножение соответствующих пар элементов и последующее суммирование полученных произведений. В случае несоответствия размеров выбрасывается исключение.

*Умножение матрицы на вектор:*

Данный алгоритм берет вектор и умножает его на матрицу. Для каждой строки матрицы вычисляется скалярное произведение на вектор, что дает новый вектор с числом элементов, равным числу строк матрицы.

*Сложение и вычитание матриц:*

Эти операции выполняются поэлементно для двух матриц одинакового размера. Каждый элемент одной матрицы складывается или вычитается с соответствующим элементом другой матрицы.

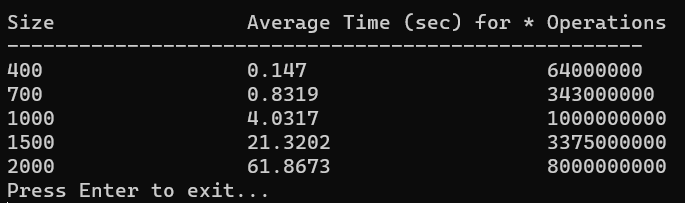
*Умножение матриц:*

Умножение матрицы на матрицу представляет собой вычисление новой матрицы , где каждый элемент равен сумме произведений элементов -й строки матрицы на соответствующие элементы -го столбца матрицы . Это требует, чтобы количество столбцов в матрице было равно количеству строк в матрице . В случае несоответствия размерностей генерируется исключение.

# Эксперименты

Эксперименты проводились на ноутбуке со следующими параметрами:

1. Операционная система: Windows 11
2. Процессор: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
3. Версия Visual Studio: 2022



В процессе анализа производительности операции умножения матриц были получены временные характеристики для различных размеров матриц. Измерения проводились в среде Visual Studio 2022 в режиме Release, что обеспечивает максимальную оптимизацию исполняемого кода. Результаты показывают, что время выполнения операции умножения матриц растет с увеличением размера матрицы, что соответствует теоретической оценке сложности алгоритма умножения матриц .

Для матрицы размером 400x400 среднее время умножения составило 0.1417 секунды, что является относительно быстрым результатом. При увеличении размера до 700x700 время увеличивается до 0.8319 секунды, что подтверждает кубическую зависимость времени от размера матрицы. Для размера 1000x1000 время выполнения возрастает до 4.0317 секунд, что существенно больше по сравнению с предыдущими размерами, отражая значительное увеличение объема вычислений. При дальнейшем увеличении размера матриц до 1500x1500 и 2000x2000 наблюдается резкое увеличение времени выполнения до 21.3202 и 61.8673 секунд соответственно. Эти результаты подтверждают, что операция умножения матриц имеет высокую вычислительную сложность, что становится особенно заметным при работе с большими матрицами.

В целом, измеренные временные характеристики соответствуют ожидаемому поведению алгоритма умножения матриц и могут быть использованы для оценки требуемых вычислительных ресурсов при работе с матричными данными большого объема.

# Заключение

В ходе выполнения данной лабораторной работы были разработаны программные средства для работы с обычными матрицами. Были созданы два основных шаблонных класса: TDynamicVector и TDynamicMatrix, которые обеспечивают не только эффективное хранение матриц, но и выполнение базовых операций над ними, включая сложение, вычитание, умножение, копирование и сравнение.

Тестовое приложение, разработанное в рамках проекта, позволяет пользователю задавать параметры матриц и выполнять над ними перечисленные операции. Это приложение демонстрирует функциональность разработанных классов.

Важной частью работы стало тестирование производительности разработанных алгоритмов. Измерения времени выполнения операции умножения матриц показали, что разработанные методы корректны и эффективны. Проведенное тестирование в режиме Release на различных размерах матриц подтвердило кубическую сложность алгоритма умножения матриц, соответствующую теоретическим предположениям.

Разработанные программные компоненты могут служить основой для дальнейшего исследования и разработки в области матричных вычислений. Их можно эффективно использовать как в учебном процессе, так и в научно-исследовательской деятельности, связанной с алгебраическими операциями над матрицами.

Таким образом, цели лабораторной работы были достигнуты, а разработанные инструменты могут быть использованы для решения широкого спектра задач, связанных с матричными операциями.

# Литература

1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# Приложение

**tmatrix.h**

// ННГУ, ИИТММ, Курс "Алгоритмы и структуры данных"

//

// Copyright (c) Сысоев А.В.

//

//

#ifndef \_\_TDynamicMatrix\_H\_\_

#define \_\_TDynamicMatrix\_H\_\_

#include <iostream>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000000;

const int MAX\_MATRIX\_SIZE = 10000;

// Динамический вектор -

// шаблонный вектор на динамической памяти

template<typename T>

class TDynamicVector

{

protected:

size\_t sz;

T\* pMem;

public:

TDynamicVector(size\_t size = 1) : sz(size)

{

if (sz <= 0 || sz > MAX\_VECTOR\_SIZE) throw out\_of\_range("Vector size should be greater than zero");

pMem = new T[sz]();// {}; // У типа T д.б. констуктор по умолчанию

}

TDynamicVector(T\* arr, size\_t s) : sz(s)

{

assert(arr != nullptr && "TDynamicVector ctor requires non-nullptr arg");

pMem = new T[sz];

std::copy(arr, arr + sz, pMem);

}

TDynamicVector(const TDynamicVector& v)

{

if (v.pMem == nullptr) {

sz = 0;

pMem = nullptr;

}

else {

sz = v.sz;

pMem = new T[sz];

std::copy(v.pMem, v.pMem + sz, pMem);

}

}

TDynamicVector(TDynamicVector&& v) noexcept // принимаем на вход r-value ссылку на объект

{

// noexcept - гарантирует, что конструктор не будет выбрасывать исключение

pMem = nullptr; // гарантируем, что при обмене с другим объектом наш pMem пустой

swap(v, \*this); // this получит ресурсы, которые ранее принадлежали объекту v, а объект v теперь будет указывать на nullptr }

~TDynamicVector()

{

if (pMem != nullptr) {

delete[] pMem;

sz = 0;

}

}

TDynamicVector& operator=(const TDynamicVector& v)

{

if (this != &v) {

if (pMem != nullptr) {

delete[] pMem;

}

if (v.pMem == nullptr) {

sz = 0;

pMem = nullptr;

}

else {

sz = v.sz;

pMem = new T[sz];

std::copy(v.pMem, v.pMem + sz, pMem);

}

}

return \*this;

}

TDynamicVector& operator=(TDynamicVector&& v) noexcept

{

if (this != &v) {

if (pMem != nullptr) {

delete[] pMem;

}

pMem = nullptr;

swap(v, \*this);

}

return \*this;

}

size\_t size() const noexcept { return sz; }

// индексация

T& operator[](size\_t ind)

{

if (ind < 0 || ind > sz) throw "error";

return pMem[ind];

}

const T& operator[](size\_t ind) const

{

if (ind < 0 || ind > sz) throw "error";

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if (pMem == nullptr) throw "Error"

if ((ind < 0) || (ind > sz)) throw "Error"

return pMem[ind];

}

const T& at(size\_t ind) const

{

if (pMem == nullptr) throw "Error"

if ((ind < 0) || (ind > sz)) throw "Error"

return pMem[ind]; }

// сравнение

bool operator==(const TDynamicVector& v) const noexcept

{

if (sz != v.sz) {

return false;

}

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

if (pMem[i] != v.pMem[i]) {

return false;

}

}

return true;

}

bool operator!=(const TDynamicVector& v) const noexcept

{

if (sz != v.sz) {

return true;

}

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

if (pMem[i] != v.pMem[i]) {

return true;

}

}

return false;

}

// скалярные операции

TDynamicVector operator+(T val)

{

TDynamicVector tmp(sz);

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

tmp.pMem[i] = pMem[i] + val;

}

return tmp;

}

TDynamicVector operator-(T val)

{

TDynamicVector tmp(sz);

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

tmp.pMem[i] = pMem[i] - val;

}

return tmp;

}

TDynamicVector operator\*(T val)

{

TDynamicVector tmp(sz);

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

tmp.pMem[i] = pMem[i] \* val;

}

return tmp;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

if (sz != v.sz) throw "error len";

TDynamicVector tmp(sz);

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

tmp.pMem[i] = pMem[i] + v.pMem[i];

}

return tmp;

}

TDynamicVector operator-(const TDynamicVector& v)

{

if (sz != v.sz) throw "error len";

TDynamicVector tmp(sz);

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

tmp.pMem[i] = pMem[i] - v.pMem[i];

}

return tmp;

}

T operator\*(const TDynamicVector& v) // noexcept(noexcept(T()))

{

if (sz != v.sz) throw "error len";

T res = T();

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

res += pMem[i] \* v.pMem[i];

}

return res;

}

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept

{

std::swap(lhs.sz, rhs.sz);

std::swap(lhs.pMem, rhs.pMem);

}

// ввод/вывод

friend istream& operator>>(istream& istr, TDynamicVector& v)

{

for (size\_t i = 0; i < v.sz; i++)

istr >> v.pMem[i]; // требуется оператор>> для типа T

return istr;

}

friend ostream& operator<<(ostream& ostr, const TDynamicVector& v)

{

for (size\_t i = 0; i < v.sz; i++)

ostr << v.pMem[i] << ' '; // требуется оператор<< для типа T

return ostr;

}

};

// Динамическая матрица -

// шаблонная матрица на динамической памяти

template<typename T>

class TDynamicMatrix : private TDynamicVector<TDynamicVector<T>>

{

using TDynamicVector<TDynamicVector<T>>::pMem;

using TDynamicVector<TDynamicVector<T>>::sz;

public:

TDynamicMatrix(size\_t s = 1) : TDynamicVector<TDynamicVector<T>>(s)

{

if (sz < 0 || sz > MAX\_MATRIX\_SIZE || sz == 0) throw "error";

for (size\_t i = 0; i < sz; i++)

pMem[i] = TDynamicVector<T>(sz);

}

using TDynamicVector<TDynamicVector<T>>::operator[];

// сравнение

bool operator==(const TDynamicMatrix& m) const noexcept

{

if (this->sz != m.sz) return false;

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

if (pMem[i] != m.pMem[i]) return false;

}

return true;

}

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix tmp(sz);

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

tmp.pMem[i] = pMem[i] \* val;

}

return tmp;

}

// матрично-векторные операции

TDynamicVector<T> operator\*(const TDynamicVector<T>& v)

{

if (sz != v.sz) throw "error len"

TDynamicVector tmp(sz);

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

T sum = T();

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

sum += pMem[i][j] \* v[j];

}

tmp[i] = sum;

}

return tmp;

}

// матрично-матричные операции

TDynamicMatrix operator+(const TDynamicMatrix& m)

{

if (sz != m.sz) throw "error len";

TDynamicMatrix tmp(sz);

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

tmp.pMem[i] = pMem[i] + m.pMem[i];

}

return tmp;

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

if (sz != m.sz) throw "error len";

TDynamicMatrix tmp(sz);

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

tmp.pMem[i] = pMem[i] - m.pMem[i];

}

return tmp;

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

if (sz != m.sz) throw "error len";

TDynamicMatrix tmp(sz);

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

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

T sum = T();

for (int k = 0; k < sz; k++) {

sum += pMem[i][k] \* m.pMem[k][j];

}

tmp.pMem[i][j] = sum;

}

}

return tmp;

}

size\_t size() const noexcept { return sz; }

// ввод/вывод

// Оператор ввода для TDynamicMatrix

friend istream& operator>>(istream& istr, TDynamicMatrix& m) {

for (size\_t i = 0; i < m.sz; i++) {

for (size\_t j = 0; j < m.pMem[i].size(); j++) {

istr >> m.pMem[i][j];

}

}

return istr;

}

// Оператор вывода для TDynamicMatrix

friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& m) {

for (size\_t i = 0; i < m.sz; i++) {

for (size\_t j = 0; j < m.pMem[i].size(); j++) {

ostr << m.pMem[i][j] << ' ';

}

ostr << endl;

}

return ostr;

}

};

#endif

**sample\_main.cpp**

// ННГУ, ИИТММ, Курс "Алгоритмы и структуры данных"

//

// Copyright (c) Сысоев А.В.

//

// Тестирование матриц

#include <iostream>

#include <limits>

#include <cstdlib>

#include<ctime>

#include<vector>

#include <iomanip>

#include "../include/tmatrix.h"

using namespace std;

int main() {

int choice;

TDynamicVector<double> vector1, vector2, vectorResult;

TDynamicMatrix<double> matrix1, matrix2, matrixResult;

setlocale(LC\_ALL, "Rus");

while (true) {

std::cout << "\n1. Создать вектор" << std::endl;

std::cout << "2. Создать матрицу" << std::endl;

std::cout << "3. Показать вектор" << std::endl;

std::cout << "4. Показать матрицу" << std::endl;

std::cout << "5. Операции над векторами" << std::endl;

std::cout << "6. Операции над матрицами" << std::endl;

std::cout << "7. Выход" << std::endl;

std::cout << "Выберите опцию: ";

std::cin >> choice;

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

switch (choice) {

case 1: {

size\_t size;

std::cout << "Введите размер вектора: ";

std::cin >> size;

vector1 = TDynamicVector<double>(size);

std::cout << "Введите элементы вектора: ";

for (size\_t i = 0; i < size; ++i) {

std::cin >> vector1[i];

}

break;

}

case 2: {

size\_t size;

std::cout << "Введите размер матрицы: ";

std::cin >> size;

matrix1 = TDynamicMatrix<double>(size);

std::cout << "Введите элементы матрицы:" << std::endl;

for (size\_t i = 0; i < size; ++i) {

for (size\_t j = 0; j < size; ++j) {

std::cin >> matrix1[i][j];

}

}

break;

}

case 3: {

std::cout << "Вектор: " << vector1 << std::endl;

break;

}

case 4: {

std::cout << "Матрица:" << std::endl << matrix1 << std::endl;

break;

}

case 5: {

int vectorChoice;

std::cout << "1. Сложение векторов" << std::endl;

std::cout << "2. Вычитание векторов" << std::endl;

std::cout << "3. Скалярное умножение векторов" << std::endl;

std::cout << "Выберите операцию: ";

std::cin >> vectorChoice;

try {

switch (vectorChoice) {

case 1: {

vector2 = TDynamicVector<double>(vector1.size());

std::cout << "Введите элементы второго вектора: ";

for (size\_t i = 0; i < vector1.size(); ++i) {

std::cin >> vector2[i];

}

vectorResult = vector1 + vector2;

std::cout << "Результат сложения: " << vectorResult << std::endl;

break;

}

default:

std::cout << "Неверный выбор операции." << std::endl;

break;

}

}

catch (const std::exception& e) {

std::cerr << "Произошла ошибка: " << e.what() << std::endl;

}

break;

}

case 6: {

int matrixChoice;

std::cout << "1. Сложение матриц" << std::endl;

std::cout << "2. Вычитание матриц" << std::endl;

std::cout << "3. Умножение матрицы на скаляр" << std::endl;

std::cout << "4. Умножение матриц" << std::endl;

std::cout << "Выберите операцию: ";

std::cin >> matrixChoice;

try {

switch (matrixChoice) {

case 1: {

matrix2 = TDynamicMatrix<double>(matrix1.size());

std::cout << "Введите элементы второй матрицы:" << std::endl;

for (size\_t i = 0; i < matrix1.size(); ++i) {

for (size\_t j = 0; j < matrix1.size(); ++j) {

std::cin >> matrix2[i][j];

}

}

matrixResult = matrix1 + matrix2;

std::cout << "Результат сложения матриц:" << std::endl << matrixResult << std::endl;

break;

}

case 2: {

matrix2 = TDynamicMatrix<double>(matrix1.size());

std::cout << "Введите элементы второй матрицы:" << std::endl;

for (size\_t i = 0; i < matrix1.size(); ++i) {

for (size\_t j = 0; j < matrix1.size(); ++j) {

std::cin >> matrix2[i][j];

}

}

matrixResult = matrix1 - matrix2;

std::cout << "Результат вычитания матриц:" << std::endl << matrixResult << std::endl;

break;

}

case 3: {

double scalar;

std::cout << "Введите скалярное значение: ";

std::cin >> scalar;

matrixResult = matrix1 \* scalar;

std::cout << "Результат умножения матрицы на скаляр:" << std::endl << matrixResult << std::endl;

break;

}

case 4: {

matrix2 = TDynamicMatrix<double>(matrix1.size());

std::cout << "Введите элементы второй матрицы для умножения:" << std::endl;

for (size\_t i = 0; i < matrix1.size(); ++i) {

for (size\_t j = 0; j < matrix1.size(); ++j) {

std::cin >> matrix2[i][j];

}

}

matrixResult = matrix1 \* matrix2;

std::cout << "Результат умножения матриц:" << std::endl << matrixResult << std::endl;

break;

}

default:

std::cout << "Неверный выбор операции." << std::endl;

break;

}

}

catch (const std::exception& e) {

std::cerr << "Произошла ошибка: " << e.what() << std::endl;

}

break;

}

case 7:

return 0;

default:

std::cout << "Неверный выбор. Попробуйте снова." << std::endl;

break;

}

std::cout << "Нажмите Enter, чтобы продолжить...";

std::cin.get();

}

return 0;

}

//int main() {

// const int sizes[] = { 400, 700, 1000, 1500, 2000 };

// const int tests = 10;

// cout << left << setw(20) << "Size"

// << setw(25) << "Average Time (sec) for \*"

// << "Operations" << endl;

// cout << "-----------------------------------------------------" << endl;

// for (int size : sizes) {

// double totalTime = 0.0;

// for (int t = 0; t < tests; ++t) {

// {

// TDynamicMatrix<double> m(size), n(size), result(size);

// for (int i = 0; i < size; ++i) {

// for (int j = 0; j < size; ++j) {

// m[i][j] = rand() % 100;

// n[i][j] = rand() % 100;

// }

// }

// clock\_t start = clock();

// result = m \* n;

// clock\_t end = clock();

// totalTime += double(end - start) / CLOCKS\_PER\_SEC;

// }

// }

// double averageTime = totalTime / tests;

// cout << setw(20) << size

// << setw(25) << averageTime

// << static\_cast<long long>(size) \* size \* size << endl;

// }

// cout << "Press Enter to exit..." << endl;

// cin.get();

// return 0;

//}