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

**Матрицы на шаблонах**

Выполнил:

студент группы 3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кондратьев Я.С.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc150686964)

[2. Постановка задачи 4](#_Toc150686965)

[3. Руководство пользователя 5](#_Toc150686966)

[4. Руководство программиста 6](#_Toc150686967)

[4.1. Описание структуры программы 6](#_Toc150686968)

[4.2. Описание структур данных 6](#_Toc150686969)

[4.2.1. TVector 6](#_Toc150686970)

[4.2.2. TMatrix (Наследуется от TVector) 7](#_Toc150686971)

[4.2.3. experiments.cpp 8](#_Toc150686972)

[4.3. Описание алгоритмов 8](#_Toc150686973)

[4.3.1. Умножение матриц: 8](#_Toc150686974)

[4.3.2. Деление матриц: 8](#_Toc150686975)

[5. Эксперименты 9](#_Toc150686976)

[6. Заключение 12](#_Toc150686977)

[7. Литература 13](#_Toc150686978)

[8. Приложение 1 14](#_Toc150686979)

[9. Приложение 2 19](#_Toc150686980)

[10. Приложение 3 22](#_Toc150686981)

[11. Приложение 4 25](#_Toc150686982)

1. Введение

В математике, матрица - это таблица, где элементы располагаются по строкам и столбцам. Внутри этой таблицы можно найти числа или математические выражения.

При работе с матрицами нужно учитывать правила матричной алгебры. Если количество строк и столбцов в матрице одинаково, она носит название квадратной. В противном случае она является прямоугольной.

В матрицах важным понятием является главная диагональ, которая включает элементы a\_11, a\_22,...,a\_nn. Самыми распространенными являются прямоугольные матрицы, которые представляют собой двумерные массивы значений, но есть и специальные виды матриц. Например, диагональная матрица - это такая матрица, где все элементы, кроме тех, что находятся на главной диагонали, равны нулю; нижнетреугольная матрица - это матрица, где все элементы выше главной диагонали также равны нулю.

Матрицу также можно рассматривать как набор векторов. Вектор в математике - это последовательность элементов, которые упорядочены определенным образом. Если рассматривать матрицу с m строками и n столбцами как набор векторов, то количество векторов будет соответствовать числу столбцов n, и каждый вектор будет содержать m элементов, соответствующих строкам матрицы. Таким образом, каждый столбец матрицы представляет собой вектор, состоящий из элементов каждой строки.

1. Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц и выполнение основных операций над ними:

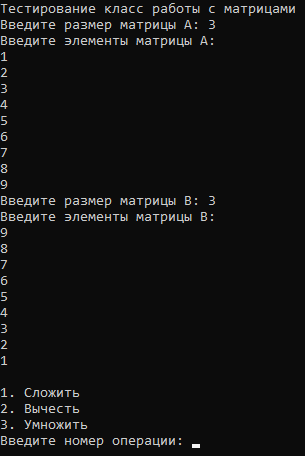
1. сложение/вычитание;
2. копирование;
3. сравнение.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

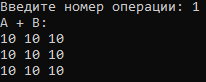
Выполнение работы предполагает решение следующих задач:

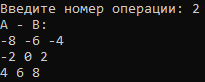
1. Реализация методов шаблонного класса TVector согласно заданному интерфейсу.
2. Реализация методов шаблонного класса TMatrix согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы классов TVector и TMatrix.
5. Модификация примера использования в тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними.
6. Руководство пользователя

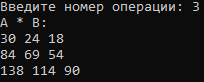
При запуске программы необходимо ввести размер первой матрицы и её значения, а также размер и значения второй матрицы. После чего выбрать операцию над введенными матрицами.

  
Рис. 1 – Исходное состояние

После выбора операции будет выведен результат и программа завершит выполнение.

  
Рис. 2 – Сумма матриц

  
Рис. 3 – Вычитание матриц

  
Рис. 4 – Умножение матриц

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль utmatirx, содержащий реализацию классов Вектор и Матрица (файл ./include/utmatrix.h).
* Модуль Test. Набор тестов для обоих классов. Включает в себя файлы test\_tvector.cpp и test\_tmatrix.cpp. Реализованы они с помощью использования фреймворка Google Test.
* Пример использования класса Матрица (файл ./samples/sample\_matrix.cpp).

## Описание структур данных

### TVector

const size\_t MAX\_VECTOR\_SIZE = 100000000 – глобальная константа, максимальный размер вектора

size\_t sz – поле класса, размер вектора

T\* pMem – динамический массив типа T

TDynamicVector(size\_t size = 1) – конструктор по умолчанию

TDynamicVector(T\* arr, size\_t s) – конструктор на основе существующего массива

TDynamicVector(const TDynamicVector& v) – конструктор копирования

TDynamicVector(TDynamicVector&& v) noexcept – конструктор перемещения

~TDynamicVector() - деструктор

TDynamicVector& operator=(const TDynamicVector& v) – оператор присваивания

TDynamicVector& operator=(TDynamicVector&& v) noexcept – оператор перемещения

size\_t size() const noexcept – получение размера вектора

T& operator[](size\_t ind) – оператор индексации (без контроля)

const T& operator[](size\_t ind) const – константный оператор индексации (без контроля)

T& at(size\_t ind) – индексация с контролем

const T& at(size\_t ind) const – константная индексация с контролем

bool operator==(const TDynamicVector& v) const noexcept – оператор сравнения

bool operator!=(const TDynamicVector& v) const noexcept – оператор сравнения

TDynamicVector operator+(T val) – оператор сложения с константой

TDynamicVector operator-(T val) – оператор вычитания с константой

TDynamicVector operator\*(T val) – оператор умножения на константу

TDynamicVector operator+(const TDynamicVector& v) – оператор сложения векторов

TDynamicVector operator-(const TDynamicVector& v) – оператор вычитания векторов

T operator\*(const TDynamicVector& v) – оператор скалярного произведения

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept – дружественная функция swap

friend istream& operator>>(istream& istr, TDynamicVector& v) – оператор ввода

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

### TMatrix (Наследуется от TVector)

public:

static const size\_t MAX\_MATRIX\_SIZE = 10000 - Константа для максимального размера матрицы

size\_t sz - Поле класса, размер матрицы

T\*\* pMem - Динамический двумерный массив типа T

TDynamicMatrix(size\_t s = 1) - Конструктор по умолчанию

bool operator==(const TDynamicMatrix & m) const noexcept - Оператор сравнения матриц на равенство

TDynamicMatrix operator\*(const T & val) - Оператор умножения матрицы на константу

TDynamicVector<T> operator\*(const TDynamicVector<T>&v) - Оператор умножения матрицы на вектор

TDynamicMatrix operator+(const TDynamicMatrix & m) - Оператор сложения матриц

TDynamicMatrix operator/(const TDynamicMatrix & m) - Оператор вычитания матриц

TDynamicMatrix operator\*(const TDynamicMatrix & m) - Оператор умножения матриц

friend istream & operator>>(istream & istr, TDynamicMatrix & v) - Дружественная функция для ввода матрицы

friend ostream & operator<<(ostream & ostr, const TDynamicMatrix & v) - Дружественная функция для вывода матрицы

## Описание алгоритмов

### Умножение матриц

Произведение матриц и формирует матрицу . Элемент в -ой строке и -ом столбце представляет собой сумму произведений элементов -ой строки матрицы на соответствующие элементы -го столбца матрицы . Реализация перегрузки оператора умножения осуществлена через три вложенных цикла: первый для итерации по строкам первой матрицы, второй — по столбцам второй матрицы, и третий — по элементам текущего столбца второй матрицы.

### Деление матриц

Делением матриц и является такая матрица , где - обратная матрица к матрице . Обратная матрица может быть найдена с помощью алгебраических дополнений: , где  – определитель матрицы ,  – транспонированная матрица алгебраических дополнений соответствующих элементов матрицы .

, где – алгебраическое дополнение

, где – минор матрицы

, где – элемент матрицы

Определитель матрицы рассчитывается рекурсивно через разложение по строке.

, где – алгебраическое дополнение

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10
2. Процессор: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz 3.60 GHz
3. Версия Visual Studio: 2022

В таблицах ниже под временем работы подразумевается усредненное время работы, основанное на 10 последовательных запусках на различных данных, полученных генерацией псевдослучайных чисел.

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора сложения (в млс)  O(n2) |
| 1 000 | 3.2 |
| 2 000 | 13 |
| 3 000 | 22.6 |
| 4 000 | 56.8 |
| 5 000 | 87.1 |
| 9 000 | 203.5 |

Таблица 1 – Измерение работы оператора сложения

Рис. 6 – График времени работы оператора сложения

Сложность оператора сложения примерно соответствует заявленной сложности

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора умножения (в млс)  O(n3) |
| 100 | 2,4 |
| 500 | 205,6 |
| 1 000 | 2800,5 |
| 2 000 | 28765,4 |
| 3 000 | 74529,2 |

Таблица 2 – Измерение времени работы оператора умножения

Рис. 7– График времени работы оператора умножения

Сложность оператора умножения примерно соответствует заявленной сложности

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора деления (в млс) |
| 6 | 0,7 |
| 7 | 10,2 |
| 8 | 71,1 |
| 9 | 504 |
| 10 | 6874,9 |

|  |  |
| --- | --- |
| 11 | 72358,4 |

Таблица 3 – Измерение времени работы оператора деления

Рис. 8 – График времени работы оператора деления

Сложность оператора деления примерно соответствует экспоненциальной сложности

1. Заключение

В ходе выполнения данной лабораторной работы я успешно справился с поставленными задачами. Изучив разные способы хранения матриц, я выбрал наиболее эффективный вариант - представление матрицы в виде вектора векторов. Чтобы достичь этой цели, я разработал вспомогательный класс TVector и расширил его функциональность в классе TMatrix.

Для более глубокого исследования этого метода хранения матриц мне пришлось создать автоматические тесты практически вручную. Однако, это не было проделано зря, так как все тесты успешно прошли и пример использования функционирует безупречно.

Кроме того, я провел оценку производительности этого метода хранения матриц и сделал интересные выводы на основе полученных значений, которые оказались лучше, чем ожидалось.

1. Литература

Эгамов А.И., Приставченко О.В. Элементы высшей математики: Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017.

Емелин А. Действия с матрицами / Емелин А. [Электронный ресурс] // Mathprofi.ru : [сайт]. — URL: <http://www.mathprofi.ru/deistviya_s_matricami.html> (дата обращения: 12.11.2023).

1. Приложение 1

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)

throw out\_of\_range("Vector size should be greater than zero");

if (size > MAX\_VECTOR\_SIZE) throw "Size should be less";

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

{

sz = v.sz;

pMem = nullptr;

std::swap(pMem, v.pMem);

}

~TDynamicVector()

{

if(pMem != nullptr)

delete[] pMem;

pMem = nullptr;

sz = 0;

}

TDynamicVector& operator=(const TDynamicVector& v)

{

if (this == &v)

return \*this;

if (sz != v.sz)

{

if (pMem != nullptr)

delete[] pMem;

sz = v.sz;

pMem = new T[sz];

}

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

return \*this;

}

TDynamicVector& operator=(TDynamicVector&& v) noexcept

{

if (this == &v)

return \*this;

if (pMem != nullptr) {

delete[] pMem;

pMem = nullptr;

}

sz = v.sz;

std::swap(pMem, v.pMem);

return \*this;

}

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

// индексация

T& operator[](size\_t ind)

{

if (ind > sz || ind < 0) throw "Index not in range";

return pMem[ind];

}

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

{

if (ind > sz || ind < 0) throw "Index not in range";

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if (ind > sz || ind < 0) throw "Index not in range";

return pMem[ind];

}

const T& at(size\_t ind) const

{

if (ind > sz || ind < 0) throw "Index not in range";

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

{

return !(\*this == v);

}

// скалярные операции

TDynamicVector operator+(T val)

{

TDynamicVector result(sz);

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

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

}

return result;

}

TDynamicVector operator-(T val)

{

TDynamicVector result(sz);

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

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

}

return result;

}

TDynamicVector operator\*(T val)

{

TDynamicVector result(sz);

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

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

}

return result;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

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

TDynamicVector result(sz);

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

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

}

return result;

}

TDynamicVector operator-(const TDynamicVector& v)

{

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

TDynamicVector result(sz);

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

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

}

return result;

}

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

{

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

T result = T();

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

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

}

return result;

}

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 (s > MAX\_MATRIX\_SIZE) throw "So big";

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 (sz != m.sz) return false;

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

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

}

return true;

}

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

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix result(sz);

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

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

}

return result;

}

// матрично-векторные операции

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

{

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

TDynamicVector<T> result(sz);

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

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

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

}

}

return result;

}

// матрично-матричные операции

TDynamicMatrix operator+(const TDynamicMatrix& m)

{

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

TDynamicMatrix result(sz);

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

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

}

return result;

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

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

TDynamicMatrix result(sz);

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

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

}

return result;

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

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

TDynamicMatrix result(sz);

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

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

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

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

}

}

}

return result;

}

// ввод/вывод

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

{

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

istr >> v.pMem[i];

}

return istr;

}

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

{

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

ostr << v.pMem[i] << endl;

}

return ostr;

}

};

#endif

1. Приложение 2

test\_vector.cpp

#include "tmatrix.h"

#include <gtest.h>

TEST(TDynamicVector, can\_create\_vector\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TDynamicVector<int> v(5));

}

TEST(TDynamicVector, cant\_create\_too\_large\_vector)

{

ASSERT\_ANY\_THROW(TDynamicVector<int> v(MAX\_VECTOR\_SIZE + 1));

}

TEST(TDynamicVector, throws\_when\_create\_vector\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TDynamicVector<int> v(-5));

}

TEST(TDynamicVector, can\_create\_copied\_vector)

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(TDynamicVector<int> v1(v));

}

TEST(TDynamicVector, copied\_vector\_is\_equal\_to\_source\_one)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(v);

EXPECT\_EQ(v, v1);

}

TEST(TDynamicVector, copied\_vector\_has\_its\_own\_memory)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(v);

v1[1] = 100;

EXPECT\_NE(v[1], v1[1]);

}

TEST(TDynamicVector, can\_get\_size)

{

TDynamicVector<int> v(4);

EXPECT\_EQ(4, v.size());

}

//TEST(TDynamicVector, can\_set\_and\_get\_element)

//{

// TDynamicVector<int> v(4);

// v[0] = 4;

//

// EXPECT\_EQ(4, v[0]);

//}

TEST(TDynamicVector, throws\_when\_set\_element\_with\_negative\_index)

{

TDynamicVector<int> v(10);

ASSERT\_ANY\_THROW(v[-1]);

}

TEST(TDynamicVector, throws\_when\_set\_element\_with\_too\_large\_index)

{

TDynamicVector<int> v(10);

ASSERT\_ANY\_THROW(v[11]);

}

TEST(TDynamicVector, can\_assign\_vector\_to\_itself)

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(v = v);

}

TEST(TDynamicVector, can\_assign\_vectors\_of\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(10);

ASSERT\_NO\_THROW(v = v1);

}

TEST(TDynamicVector, assign\_operator\_change\_vector\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

v = v1;

EXPECT\_EQ(v.size(), 11);

}

TEST(TDynamicVector, can\_assign\_vectors\_of\_different\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

ASSERT\_NO\_THROW(v = v1);

}

TEST(TDynamicVector, compare\_equal\_vectors\_return\_true)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(10);

EXPECT\_EQ(v == v1, true);

}

TEST(TDynamicVector, compare\_vector\_with\_itself\_return\_true)

{

TDynamicVector<int> v(10);

EXPECT\_EQ(v == v, true);

}

TEST(TDynamicVector, vectors\_with\_different\_size\_are\_not\_equal)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

EXPECT\_NE(true, v == v1);

}

TEST(TDynamicVector, can\_add\_scalar\_to\_vector)

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(v + 1);

}

TEST(TDynamicVector, can\_subtract\_scalar\_from\_vector)

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(v - 1);

}

TEST(TDynamicVector, can\_multiply\_scalar\_by\_vector)

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(v \* 2);

}

TEST(TDynamicVector, can\_add\_vectors\_with\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(10);

ASSERT\_NO\_THROW(v + v1);

}

TEST(TDynamicVector, cant\_add\_vectors\_with\_not\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

ASSERT\_ANY\_THROW(v + v1);

}

TEST(TDynamicVector, can\_subtract\_vectors\_with\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(10);

ASSERT\_NO\_THROW(v - v1);

}

TEST(TDynamicVector, cant\_subtract\_vectors\_with\_not\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

ASSERT\_ANY\_THROW(v - v1);

}

TEST(TDynamicVector, can\_multiply\_vectors\_with\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(10);

ASSERT\_NO\_THROW(v \* v1);

}

TEST(TDynamicVector, cant\_multiply\_vectors\_with\_not\_equal\_size)

{

TDynamicVector<int> v(10);

TDynamicVector<int> v1(11);

ASSERT\_ANY\_THROW(v \* v1);

}

1. Приложение 3

test\_matrix.cpp

#include "tmatrix.h"

#include <gtest.h>

TEST(TDynamicMatrix, can\_create\_matrix\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TDynamicMatrix<int> m(5));

}

TEST(TDynamicMatrix, cant\_create\_too\_large\_matrix)

{

ASSERT\_ANY\_THROW(TDynamicMatrix<int> m(MAX\_MATRIX\_SIZE + 1));

}

TEST(TDynamicMatrix, throws\_when\_create\_matrix\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TDynamicMatrix<int> m(-5));

}

TEST(TDynamicMatrix, can\_create\_copied\_matrix)

{

TDynamicMatrix<int> m(5);

ASSERT\_NO\_THROW(TDynamicMatrix<int> m1(m));

}

TEST(TDynamicMatrix, copied\_matrix\_is\_equal\_to\_source\_one)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(10);

EXPECT\_EQ(m, m1);

}

TEST(TDynamicMatrix, copied\_matrix\_has\_its\_own\_memory)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(m);

m1[1][1] = 1;

EXPECT\_NE(m[1][1], m1[1][1]);

}

TEST(TDynamicMatrix, can\_get\_size)

{

TDynamicMatrix<int> m(10);

EXPECT\_EQ(10, m.size());

}

TEST(TDynamicMatrix, can\_set\_and\_get\_element)

{

TDynamicMatrix<int> m(10);

m[1][1] = 5;

EXPECT\_EQ(5, m[1][1]);

}

TEST(TDynamicMatrix, throws\_when\_set\_element\_with\_negative\_index)

{

TDynamicMatrix<int> m(10);

ASSERT\_ANY\_THROW(m[1][-1] = 5);

}

TEST(TDynamicMatrix, throws\_when\_set\_element\_with\_too\_large\_index)

{

TDynamicMatrix<int> m(10);

ASSERT\_ANY\_THROW(m[1][11] = 5);

}

TEST(TDynamicMatrix, can\_assign\_matrix\_to\_itself)

{

TDynamicMatrix<int> m(10);

ASSERT\_NO\_THROW(m = m);

}

TEST(TDynamicMatrix, can\_assign\_matrices\_of\_equal\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(10);

ASSERT\_NO\_THROW(m = m1);

}

TEST(TDynamicMatrix, assign\_operator\_change\_matrix\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(11);

m = m1;

EXPECT\_EQ(m.size(), 11);

}

TEST(TDynamicMatrix, can\_assign\_matrices\_of\_different\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(11);

ASSERT\_NO\_THROW(m = m1);

}

TEST(TDynamicMatrix, compare\_equal\_matrices\_return\_true)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(m);

EXPECT\_EQ(m == m1, true);

}

TEST(TDynamicMatrix, compare\_matrix\_with\_itself\_return\_true)

{

TDynamicMatrix<int> m(10);

EXPECT\_EQ(m == m, true);

}

TEST(TDynamicMatrix, matrices\_with\_different\_size\_are\_not\_equal)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(11);

EXPECT\_NE(m == m1, true);

}

TEST(TDynamicMatrix, can\_add\_matrices\_with\_equal\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(10);

ASSERT\_NO\_THROW(m + m1);

}

TEST(TDynamicMatrix, cant\_add\_matrices\_with\_not\_equal\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(11);

ASSERT\_ANY\_THROW(m + m1);

}

TEST(TDynamicMatrix, can\_subtract\_matrices\_with\_equal\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(10);

ASSERT\_NO\_THROW(m - m1);

}

TEST(TDynamicMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size)

{

TDynamicMatrix<int> m(10);

TDynamicMatrix<int> m1(11);

ASSERT\_ANY\_THROW(m - m1);

}

1. Приложение 4

sample\_matrix.cpp

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

//

// Copyright (c) Сысоев А.В.

//

// Тестирование матриц

#include <iostream>

#include "tmatrix.h"

//---------------------------------------------------------------------------

void main()

{

setlocale(LC\_ALL, "Russian");

cout << "Тестирование класс работы с матрицами" << endl;

int size;

cout << "Введите размер матрицы A: ";

cin >> size;

TDynamicMatrix<int> a(size);

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

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

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

{

cin >> a[i][j];

}

cout << "Введите размер матрицы B: ";

cin >> size;

TDynamicMatrix<int> b(size);

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

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

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

{

cin >> b[i][j];

}

int act;

cout << endl << "1. Сложить"<<endl<< "2. Вычесть"<<endl << "3. Умножить"<< endl;

cout << "Введите номер операции: ";

cin >> act;

switch (act)

{

case 1:

cout << "A + B:" << endl << a + b << endl;

break;

case 2:

cout << "A - B:" << endl << a - b << endl;

break;

case 3:

cout << "A \* B:" << endl << a \* b << endl;

break;

default:

break;

}

}

//---------------------------------------------------------------------------