МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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»

Профиль подготовки: «Общий»

**Отчет по лабораторной работе**

на тему:

**«Структуры хранения матриц специального вида»**

**Выполнил:** студент группы 3822Б1ПР2

Чернова Наталья Алексеевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород  
2023

**Содержание**

[1. ВВЕДЕНИЕ 3](#_Toc4)

[2. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc5)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 5](#_Toc6)

[4. РУКОВОДСТВО ПРОГРАММИСТА 8](#_Toc7)

[4.1 Описание структуры программы 8](#_Toc8)

[4.2 Описание структур данных 8](#_Toc9)

[5. ОПИСАНИЕ АЛГОРИТМОВ 10](#_Toc10)

[6. ЭКСПЕРИМЕНТЫ 11](#_Toc11)

[7. ЗАКЛЮЧЕНИЕ 12](#_Toc12)

[8. ЛИТЕРАТУРА 13](#_Toc13)

[9. ПРИЛОЖЕНИЕ 14](#_Toc14)

# 1. ВВЕДЕНИЕ

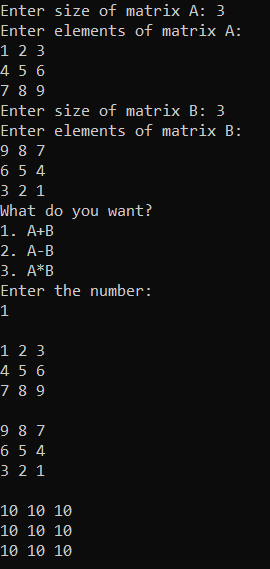
Матрица – это математический объект, записываемый в виде прямоугольной таблицы элементов, который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Элементами матрицы являются числа или выражения. Так же матрицу можно представить как набор векторов. Вектор – это последовательность однородных элементов. Если у нас есть матрица размера mn, то мы можем считать, что у нас есть m векторов размера n. Матрицы можно складывать (если они имеют одинаковый размер), умножать на число, на вектор или друг на друга. Также матрицы используют для записи систем алгебраических или дифференциальных уравнений. В данной лабораторной работе будет выполнена реализация динамических шаблонных классов вектора и матрицы, и реализация проверки данных классов с использованием фреймворка GoogleTest.

# 2. ПОСТАНОВКА ЗАДАЧИ

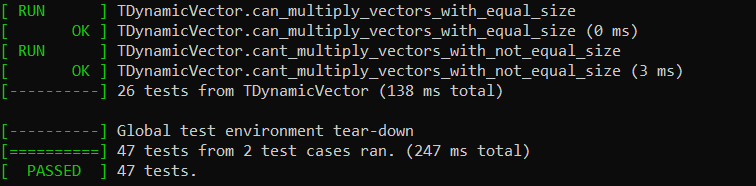
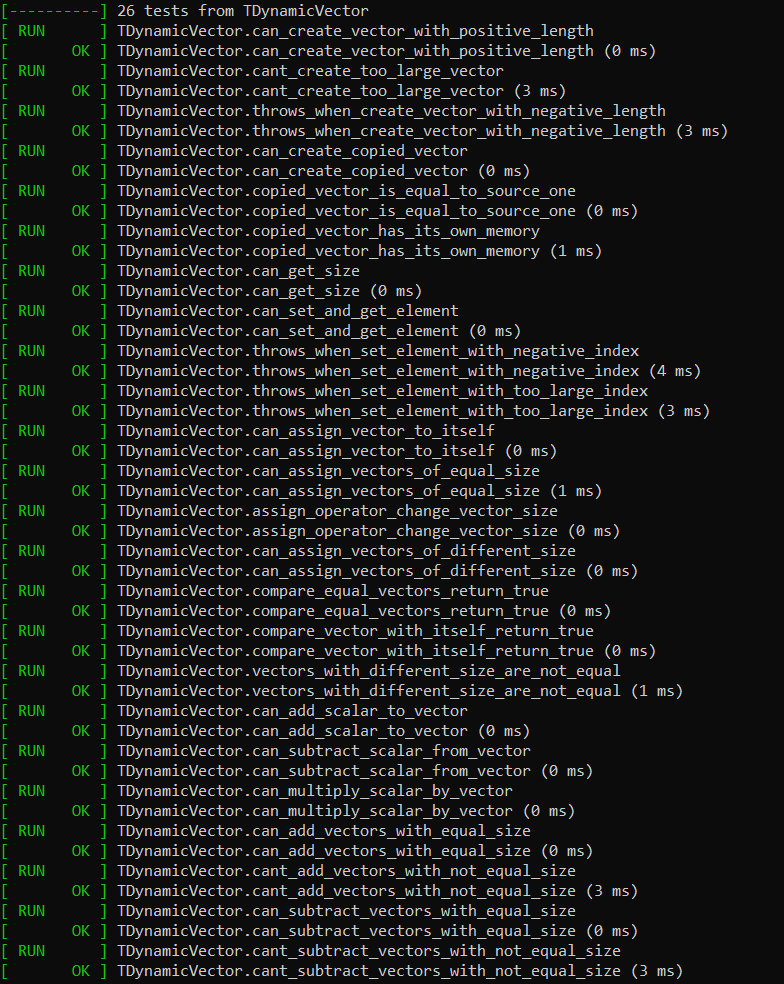
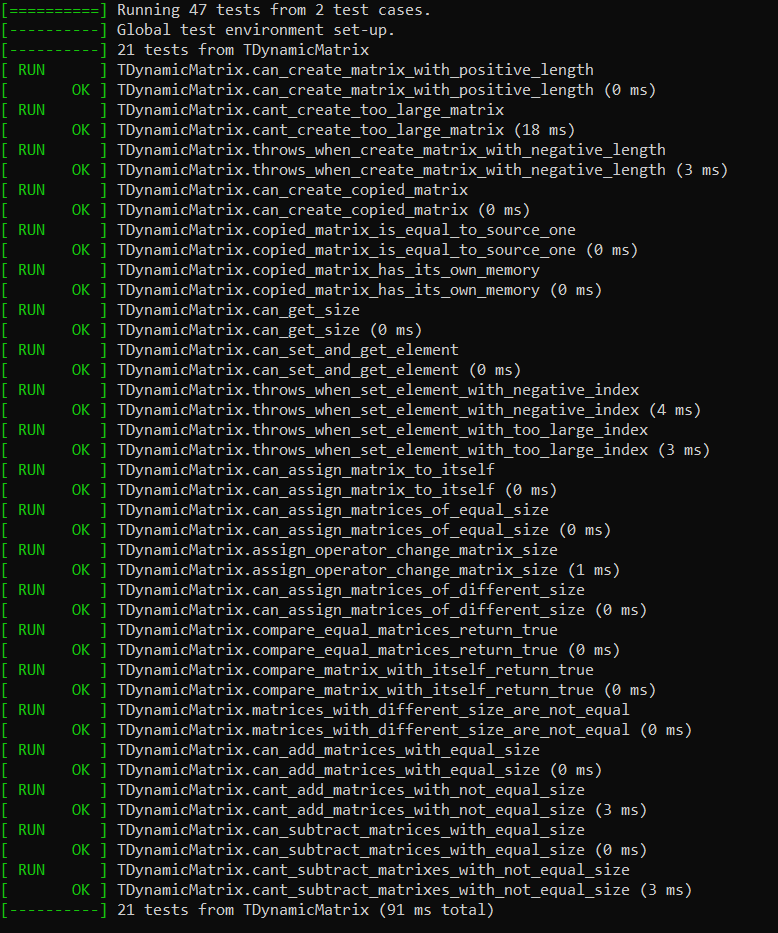
Задача данной работы – реализовать шаблонные классы вектора и матрицы, а также реализация проверки данных классов с использованием фреймворка GoogleTest.

# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске файла sample\_matrix.cpp будет выведено окно, которое предлагает ввести размеры двух матриц и их элементы. После этого будет выведен выбор операций над матрицами. После того, как пользователь сделает выбор, будет произведена выбранная операция и выведены на экран все результаты.



При запуске проекта test\_utmatrix будут выведены результаты работы тестов, проверяющих корректность написанных классов.



# 4. РУКОВОДСТВО ПРОГРАММИСТА

## 4.1 Описание структуры программы

Программа состоит из нескольких проектов:

* Проект sample\_matrix, содержащий файл tmatrix.h, где находится реализация классов вектора и матрицы
* Проект test\_utmatrix, содержащий тесты для проверки корректности работы классов вектора и матрицы. Реализованы с помощью фреймворка GoogleTest.

## 4.2 Описание структур данных

TDynamicVector (шаблонный класс)

protected:

size\_t sz; //размер вектора

T\* pMem; //динамический массив типа T

public:

TDynamicVector(size\_t size = 1) //конструктор

TDynamicVector(T\* arr, size\_t s) //конструктор

TDynamicVector(const TDynamicVector& v) //конструктор копирования

TDynamicVector(TDynamicVector&& v) //конструктор перемещения

~TDynamicVector() //деструктор

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

TDynamicVector& operator=(TDynamicVector&& v) //оператор перемещения

size\_t size() //получение размера вектора

T& operator[](size\_t ind)// индексация

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

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

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

bool operator==(const TDynamicVector& v) //оператор сравнения

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

TDynamicVector operator+(T val) //скалярное сложение

TDynamicVector operator-(T val) //скалярное вычитание

TDynamicVector operator\*(T val) //скалярное умножение

TDynamicVector operator+(const TDynamicVector& v) //векторное сложение

TDynamicVector operator-(const TDynamicVector& v) //векторное вычитание

T operator\*(const TDynamicVector& v) //векторное умножение

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) //обмен элементов

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

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

TDynamicMatrix (шаблонный класс, наследующий класс TDynamicVector)

private:

using TDynamicVector<TDynamicVector<T>>::pMem; //динамический массив типа T

using TDynamicVector<TDynamicVector<T>>::sz; //размер матрицы

public:

using TDynamicVector<TDynamicVector<T>>::size; //получение размера

using TDynamicVector<TDynamicVector<T>>::at; //индексация с контролем

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

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

bool operator==(const TDynamicMatrix& m) const //оператор сравнения

TDynamicMatrix operator\*(const T& val) //умножение матрицы на число

TDynamicVector<T> operator\*(const TDynamicVector<T>& v) //умножение вектора на вектор

TDynamicMatrix operator+(const TDynamicMatrix& m) //сложение матриц

TDynamicMatrix operator-(const TDynamicMatrix& m) //вычитание матриц

TDynamicMatrix operator\*(const TDynamicMatrix& m) //умножение матриц

friend istream& operator>>(istream& istr, TDynamicMatrix& m) //оператор ввода

friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& m) //оператор вывода

# 5. ОПИСАНИЕ АЛГОРИТМОВ

Сложение и вычитание матриц не являются сложными операциями. Для этого необходимо просто сложить или вычесть элементы с одинаковыми индексами. А вот умножение матриц высчитывается иным способом. Матрицы могут быть перемножены только в том случае, если число столбцов первой матрицы равно числу строк второй матрицы (такие матрицы называются совместимыми). Каждый элемент результирующей матрицы является суммой произведений каждого члена строки первой матрицы на каждый член столбца второй матрицы с соответствующими индексами. Например, если мы умножаем матрицы A(km) и B(mn), то результатом будет матрица C(kn), где .

# 6. ЭКСПЕРИМЕНТЫ

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10

2. Процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 2100 МГц

3. Версия Visual Studio: 2022

|  |  |
| --- | --- |
| Кол-во элементов (n) | Время работы оператора сложения (ms) |
| 1000 | 16 |
| 3000 | 130 |
| 6000 | 573 |

Здесь представлено время работы проекта, высчитывающего сумму двух матриц, размером nn элементов. Сложность оператора примерно равно O(n2).

# 7. ЗАКЛЮЧЕНИЕ

В ходе работы были реализованы шаблонные классы вектора и матрицы, а также тесты для проверки корректности работы данных классов.

# 8. ЛИТЕРАТУРА

[Матрица (математика) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))

[Умножение матриц — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86)

[Pract\_ADS.pdf - Google Диск](https://drive.google.com/file/d/1aZEfnRSgA7IhMccdLTvbJ5noZlXPC1BK/view)

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.

# 9. 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)

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

if (sz > MAX\_VECTOR\_SIZE)

throw out\_of\_range("This is too much");

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

{

pMem = nullptr;

swap(v, \*this);

}

~TDynamicVector()

{

if (pMem != nullptr) {

delete[]pMem;

sz = 0;

}

}

TDynamicVector& operator=(const TDynamicVector& v)

{

if (pMem != nullptr) {

delete[]pMem;

sz = 0;

}

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 (&v != this) {

if (pMem != nullptr) {

delete[]pMem;

sz = 0;

}

sz = v.sz;

pMem = v.pMem;

v.sz = 0;

v.pMem = nullptr;

}

else {

throw "This is the same";

}

return \*this;

}

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

// индексация

T& operator[](size\_t ind)

{

if ((ind < 0) || (ind > sz)) {

throw "out of range";

}

if (pMem == nullptr) {

throw "It's empty";

}

return pMem[ind];

}

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

{

if ((ind < 0) || (ind > sz)) {

throw "out of range";

}

if (pMem == nullptr) {

throw "It's empty";

}

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if ((ind < 0) || (ind > sz)) {

throw "out of range";

}

if (pMem == nullptr) {

throw "It's empty";

}

return pMem[ind];

}

const T& at(size\_t ind) const

{

if ((ind < 0) || (ind > sz)) {

throw "out of range";

}

if (pMem == nullptr) {

throw "It's empty";

}

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 tdv(\*this);

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

tdv.pMem[i] += val;

}

return tdv;

}

TDynamicVector operator-(T val)

{

TDynamicVector tdv(\*this);

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

tdv.pMem[i] -= val;

}

return tdv;

}

TDynamicVector operator\*(T val)

{

TDynamicVector tdv(\*this);

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

tdv.pMem[i] \*= val;

}

return tdv;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

if (sz != v.sz) {

throw "different sizes";

}

TDynamicVector tdv(\*this);

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

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

}

return tdv;

}

TDynamicVector operator-(const TDynamicVector& v)

{

if (sz != v.sz) {

throw "different sizes";

}

TDynamicVector tdv(\*this);

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

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

}

return tdv;

}

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

{

if (sz != v.sz) {

throw "different sizes";

}

T scalar = 0;

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

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

}

return scalar;

}

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:

using TDynamicVector<TDynamicVector<T>>::size;

using TDynamicVector<TDynamicVector<T>>::at;

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

{

if (sz > MAX\_MATRIX\_SIZE)

throw out\_of\_range("This is too much");

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;

}

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix tdm(\*this);

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

tdm.pMem[i] \*= val;//check

}

return tdm;

////////////////

}

// матрично-векторные операции

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

{

if (sz != v.sz) {

throw "different sizes";

}

TDynamicVector<T> tdv(v.sz);

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

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

//tdv.pMem[i] += pMem[k] \* m.pMem[k];

tdv[i] = tdv[i] + pMem[i][k] + v[k];

}

}

return tdv;

}

// матрично-матричные операции

TDynamicMatrix operator+(const TDynamicMatrix& m)

{

if (sz != m.sz) {

throw "different sizes";

}

TDynamicMatrix tdm(\*this);

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

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

}

return tdm;

////////////////

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

if (sz != m.sz) {

throw "different sizes";

}

TDynamicMatrix tdm(\*this);

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

tdm.pMem[i] = pMem[i] - m.pMem[i];//check

}

return tdm;

////////////////

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

TDynamicMatrix tdm(\*this);

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

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

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

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

}

}

}

return tdm;

////////////////

}

// ввод/вывод

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

{

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

istr >> m.pMem[i];

return istr;

}

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

{

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

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

return ostr;

}

};

#endif

**sample\_matrix.cpp**

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

//

// Copyright (c) Сысоев А.В.

//

// Тестирование матриц

#include <iostream>

#include <ctime>

#include "tmatrix.h"

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

void main()

{

/\*

TDynamicMatrix<int> a(5), b(5), c(5);

int i, j;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование класс работы с матрицами"

<< endl;

for (i = 0; i < 5; i++)

for (j = 0; j < 5; j++)

{

a[i][j] = i \* 10 + j;

b[i][j] = (i \* 10 + j) \* 100;

}

c = a + b;

cout << "Matrix a = " << endl << a << endl;

cout << "Matrix b = " << endl << b << endl;

cout << "Matrix c = a + b" << endl << c << endl;

\*/

int na, nb;

cout << "Enter size of matrix A: ";

cin >> na;

TDynamicMatrix<int> A(na);

cout << "Enter elements of matrix A: ";

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

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

cin >> A[i][j];

}

}

cout << "Enter size of matrix B: ";

cin >> nb;

TDynamicMatrix<int> B(nb);

cout << "Enter elements of matrix B: ";

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

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

cin >> B[i][j];

}

}

TDynamicMatrix<int> C(nb);

int f;

cout << "What do you want?" << endl;

cout << "1. A+B" << endl;

cout << "2. A-B" << endl;

cout << "3. A\*B" << endl;

cout << "Enter the number: " << endl;

cin >> f;

switch (f) {

case 1:

C = A + B;

break;

case 2:

C = A - B;

break;

case 3:

cout << A \* B;

C = A \* B;

break;

default:

cout << "ERROR";

break;

}

cout << endl << A << endl << B << endl << C << endl;

/\*

int n = 6000;

TDynamicMatrix<int> m1(n), m2(n), m3(n);

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

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

m1[i][j] = i;

m2[i][j] = i;

}

}

unsigned int start\_time = clock();

m3 = m1 + m2;

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

cout << search\_time;

\*/

}

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

**test\_tvector.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(5);

v[2] = 5;

TDynamicVector<int> v1(v);

EXPECT\_EQ(5, v1[2]);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(v);

EXPECT\_NE(&v, &v1);

}

TEST(TDynamicVector, can\_get\_size)

{

TDynamicVector<int> v(5);

EXPECT\_EQ(5, v.size());

}

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

{

TDynamicVector<int> v(5);

v[0] = 4;

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

}

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

{

TDynamicVector<int> v(5);

ASSERT\_ANY\_THROW(v[-5]);

}

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

{

TDynamicVector<int> v(5);

ASSERT\_ANY\_THROW(v[10]);

}

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

{

TDynamicVector<int> v(5);

ASSERT\_NO\_THROW(v = v);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(5);

ASSERT\_NO\_THROW(v = v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

v1 = v;

EXPECT\_EQ(5, v1.size());

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

ASSERT\_NO\_THROW(v = v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(5);

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

}

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

{

TDynamicVector<int> v(5);

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

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

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

}

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

{

TDynamicVector<int> v(5);

int a = 2;

ASSERT\_NO\_THROW(v + a);

}

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

{

TDynamicVector<int> v(5);

int a = 2;

ASSERT\_NO\_THROW(v - a);

}

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

{

TDynamicVector<int> v(5);

int a = 2;

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

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(5);

ASSERT\_NO\_THROW(v + v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

ASSERT\_ANY\_THROW(v + v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(5);

ASSERT\_NO\_THROW(v - v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

ASSERT\_ANY\_THROW(v - v1);

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(5);

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

}

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

{

TDynamicVector<int> v(5);

TDynamicVector<int> v1(6);

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

}

**test\_tmatrix.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(5);

m[2][1] = 5;

TDynamicMatrix<int> m1(m);

EXPECT\_EQ(5, m1[2][1]);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(m);

EXPECT\_NE(&m, &m1);

}

TEST(TDynamicMatrix, can\_get\_size)

{

TDynamicMatrix<int> m(5);

EXPECT\_EQ(5, m.size());

}

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

{

TDynamicMatrix<int> m(5);

m[1] = 2;

ASSERT\_NO\_THROW(m.at(1));

}

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

{

TDynamicMatrix<int> m(5);

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

}

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

{

TDynamicMatrix<int> m(5);

ASSERT\_ANY\_THROW(m[10]);

}

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

{

TDynamicMatrix<int> m(5);

ASSERT\_NO\_THROW(m = m);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(5);

ASSERT\_NO\_THROW(m = m1);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(6);

m1 = m;

EXPECT\_EQ(5, m1.size());

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(6);

ASSERT\_NO\_THROW(m = m1);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(5);

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

}

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

{

TDynamicMatrix<int> m(5);

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

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(6);

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

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(5);

ASSERT\_NO\_THROW(m + m1);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(6);

ASSERT\_ANY\_THROW(m + m1);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(5);

ASSERT\_NO\_THROW(m - m1);

}

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

{

TDynamicMatrix<int> m(5);

TDynamicMatrix<int> m1(6);

ASSERT\_ANY\_THROW(m - m1);

}

**test\_main.cpp**

#include <gtest.h>

int main(int argc, char \*\*argv)

{

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}