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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ермолаев В.А.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc150686964)

[2. Постановка задачи 4](#_Toc150686965)

[3. Руководство пользователя 5](#_Toc150686966)

[4. Руководство программиста 7](#_Toc150686967)

[4.1. Описание структуры программы 7](#_Toc150686968)

[4.2. Описание структур данных 7](#_Toc150686969)

[4.2.1. TVector 7](#_Toc150686970)

[4.2.2. TMatrix (Наследуется от TVector) 8](#_Toc150686971)

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

[4.3. Описание алгоритмов 9](#_Toc150686973)

[4.3.1. Умножение матриц: 9](#_Toc150686974)

[4.3.2. Деление матриц: 9](#_Toc150686975)

[5. Эксперименты 10](#_Toc150686976)

[6. Заключение 13](#_Toc150686977)

[7. Литература 14](#_Toc150686978)

[8. Приложение 1 15](#_Toc150686979)

[9. Приложение 2 22](#_Toc150686980)

[10. Приложение 3 25](#_Toc150686981)

[11. Приложение 4 28](#_Toc150686982)

[12. Приложение 5 29](#_Toc150686983)

1. Введение

Матрица в области математики - это таблица, состоящая из элементов, расположенных по строкам и столбцам. В этой таблице могут быть числа или математические выражения.

Для операций с матрицами необходимо учитывать правила матричной алгебры. Если матрица имеет одинаковое количество строк и столбцов, она называется квадратной матрицей, иначе - прямоугольной.

Важным понятием в матрицах является главная диагональ, которая включает элементы . Чаще всего используются прямоугольные матрицы, представляющие из себя двумерный массив значений, но кроме них есть также и специальные виды матриц. Например, диагональная матрица - это матрица, в которой все элементы, не принадлежащие главной диагонали, равны нулю; нижнетреугольная матрица - матрица, в которой все элементы, находящиеся выше главной диагонали, равны нулю.

Матрицу также можно рассматривать как набор векторов. Вектор в математике представляет собой последовательность элементов, упорядоченных определенным образом. Если матрицу с строками и столбцами рассматривать как ряд векторов, то число векторов будет соответствовать числу столбцов , и каждый вектор будет содержать 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.

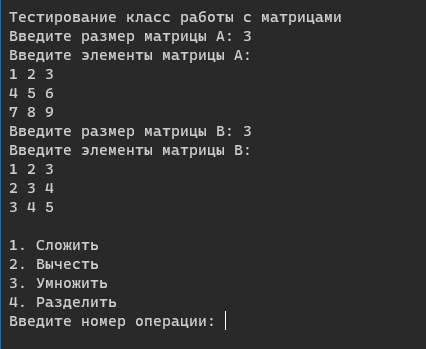


Рис. 1 – Исходное состояние

После выбора операции будет выведен результат и программа завершит выполнение.

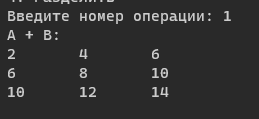


Рис. 2 – Сумма матриц

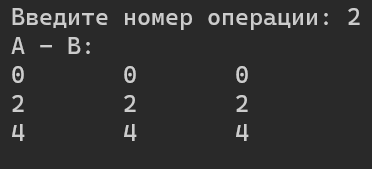


Рис. 3 – Вычитание матриц

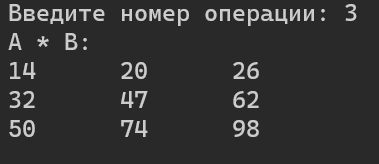


Рис. 4 – Умножение матриц

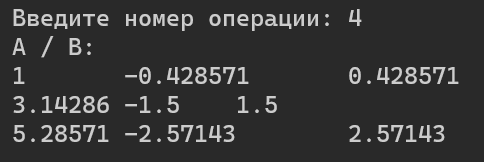


Рис. 5 – Деление матриц

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль utmatirx, содержащий реализацию классов Вектор и Матрица (файл ./include/utmatrix.h).
* Модули VectorTest и MatrixTest. 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/(T val) – оператор деления 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

TDynamicMatrix operator/(const T & val) - Оператор деления 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

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

T getDeterminant() - Метод для получения определителя матрицы

TDynamicMatrix getTransposedMatrix() - Метод для получения транспонированной матрицы

TDynamicMatrix getInverseMatrix() - Метод для получения обратной матрицы

T & at(size\_t i, size\_t j) - Метод индексации с контролем

const T & at(size\_t i, size\_t j) const - Константный метод индексации с контролем

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

private:

static T calcDeterminant(TDynamicMatrix<T>&m) - Статический метод для вычисления определителя матрицы

TDynamicMatrix getMinor(size\_t i, size\_t j) - Метод для получения минора матрицы

### experiments.cpp

enum operations {

PLUS,

MINUS,

MULTIPLY,

DIVIDE,

COPY, COMPARE}; - перечисление констант для запуска тестов производительности

## 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 11
2. Процессор: Intel(R) Core™ i3-1115G4- CPU @ 3.00 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.

Таблица 1 – Измерение работы оператора сложения

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора сложения (в млс)  O(n2) |
| 1 000 | 6.4 |
| 2 000 | 25 |
| 3 000 | 56.2 |
| 4 000 | 106.4 |
| 5 000 | 216.2 |
| 9 000 | 516.6 |



Рис. 6 – График времени работы оператора сложения

Сложность оператора сложения примерно соответствует заявленной сложности

Таблица 2 – Измерение времени работы оператора умножения

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора умножения (в млс)  O(n3) |
| 100 | 5.2 |
| 500 | 443.3 |
| 1 000 | 6036.7 |
| 2 000 | 63304.8 |
| 3 000 | 190328.0 |

Рис. 7– График времени работы оператора умножения

Сложность оператора умножения примерно соответствует заявленной сложности

Таблица 3 – Измерение времени работы оператора деления

|  |  |
| --- | --- |
| Количество элементов | Время работы оператора деления (в млс) |
| 6 | 1.8 |
| 7 | 21.5 |
| 8 | 135.2 |
| 9 | 1179 |
| 10 | 14486.2 |

Продолжение таблицы №3

|  |  |
| --- | --- |
| 11 | 147455.0 |



Рис. 8 – График времени работы оператора деления

Сложность оператора деления примерно соответствует экспоненциальной сложности

1. Заключение

В рамках данной лабораторной работы я успешно справился с поставленными задачами. Изучив различные методы хранения матриц, я выбрал один из самых эффективных вариантов - представление матрицы в виде вектора векторов. Для достижения этой цели я разработал вспомогательный класс TVector и расширил его функционал в классе TMatrix.

Для углубленного исследования данного метода хранения матриц мне пришлось создать автоматические тесты практически вручную. Однако, это оказалось не напрасной работой, поскольку все тесты успешно проходят и пример использования функционирует безупречно.

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

1. Литература
2. Эгамов А.И., Приставченко О.В. Элементы высшей математики: Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017.
3. 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).
4. Приложение 1

tmatrix.h

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

//

// Copyright (c) Сысоев А.В.

//

//

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

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

#include <iostream>

#include <cassert>

#include <cmath>

using namespace std;

const size\_t MAX\_VECTOR\_SIZE = 100000000;

const size\_t 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("Size should be greater than zero");

if (sz > MAX\_VECTOR\_SIZE)

throw length\_error("Exceeded vector max size");

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");

if (sz > MAX\_VECTOR\_SIZE)

throw length\_error("Exceeded vector max size");

pMem = new T[sz];

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

}

TDynamicVector(const TDynamicVector& v)

{

sz = v.sz;

pMem = new T[sz];

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

{

return pMem[ind];

}

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

{

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if (ind >= sz)

throw out\_of\_range("Index shoud be less then size");

return pMem[ind];

}

const T& at(size\_t ind) const

{

if (ind >= sz)

throw out\_of\_range("Index shoud be less then size");

return pMem[ind];

}

// сравнение

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

{

if (sz != v.sz)

return false;

for (size\_t 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(\*this);

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

result.pMem[i] += val;

return result;

}

TDynamicVector operator-(T val)

{

TDynamicVector result(\*this);

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

result.pMem[i] -= val;

return result;

}

TDynamicVector operator\*(T val)

{

TDynamicVector result(\*this);

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

result.pMem[i] \*= val;

return result;

}

// Костыль

TDynamicVector operator/(T val)

{

TDynamicVector result(\*this);

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

result.pMem[i] /= val;

return result;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

if (sz != v.sz)

throw logic\_error("Sizes are not equal");

TDynamicVector result(sz);

for (size\_t 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 logic\_error("Sizes are not equal");

TDynamicVector result(sz);

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

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

}

return result;

}

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

{

if (sz != v.sz)

throw logic\_error("Sizes are not equal");

T result = T();

for (size\_t 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] << '\t' - 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 length\_error("Exceeded matrix max size");

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

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

}

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

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

// сравнение

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

{

if (sz != m.sz)

return false;

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

if (pMem[i] != m.pMem[i])

return false;

}

return true;

}

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix result(\*this);

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

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

}

return result;

}

TDynamicMatrix operator/(const T& val)

{

TDynamicMatrix result(\*this);

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

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

}

return result;

}

// матрично-векторные операции

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

{

if (sz != v.sz)

throw logic\_error("Sizes are not equal");

TDynamicVector<T> result(sz);

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

for (size\_t 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 logic\_error("Sizes are not equal");

TDynamicMatrix result(\*this);

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

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

return result;

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw logic\_error("Sizes are not equal");

TDynamicMatrix result(\*this);

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

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

return result;

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw logic\_error("Sizes are not equal");

TDynamicMatrix result(m.sz);

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

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

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

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

}

}

}

return result;

}

TDynamicMatrix operator/(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw logic\_error("Sizes are not equal");

TDynamicMatrix tmp(m);

return \*this \* tmp.getInverseMatrix();

}

T getDeterminant() {

return calcDeterminant(\*this);

}

TDynamicMatrix getTransposedMatrix() {

TDynamicMatrix tmp(\*this);

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

for (size\_t j = i; j < tmp.sz; j++) {

std::swap(tmp.pMem[i][j], tmp.pMem[j][i]);

}

}

return tmp;

}

TDynamicMatrix getInverseMatrix() {

TDynamicMatrix tmp(sz);

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

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

auto minor = getMinor(i, j);

tmp.pMem[i][j] = std::pow(-1, i + j) \* calcDeterminant(minor);

}

}

return tmp.getTransposedMatrix() / getDeterminant();

}

T& at(size\_t i, size\_t j) {

if (i >= sz || j >= sz)

throw out\_of\_range("Index shoud be less then size");

return pMem[i][j];

}

const T& at(size\_t i, size\_t j) const {

if (i >= sz || j >= sz)

throw out\_of\_range("Index shoud be less then size");

return pMem[i][j];

}

// ввод/вывод

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

{

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

istr >> v.pMem[i];

return istr;

}

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

{

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

ostr << v.pMem[i] << '\n';

return ostr;

}

private:

static T calcDeterminant(TDynamicMatrix<T>& m) {

if (m.sz == 1)

return m.pMem[0][0];

else if (m.sz == 2)

return m.pMem[0][0] \* m.pMem[1][1] - m.pMem[0][1] \* m.pMem[1][0];

else {

T det = T();

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

auto minor = m.getMinor(0, k);

det = det + std::pow(-1, k) \* m.pMem[0][k] \* calcDeterminant(minor);

}

return det;

}

}

TDynamicMatrix getMinor(size\_t i, size\_t j) {

if (sz == 1)

return \*this;

TDynamicMatrix minor(sz - 1);

for (size\_t i1 = 0, i2 = 0; i1 < sz && i2 < minor.sz; i1++) {

for (size\_t j1 = 0, j2 = 0; j1 < sz && j2 < minor.sz; j1++) {

if (j1 == j || i1 == i)

continue;

minor[i2++][j2++] = pMem[i1][j1];

}

}

return minor;

}

};

#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)

{

int\* arr1 = new int[3] { 1, 2, 3 };

TDynamicVector<int> v1(arr1, 3), v3(v1);

EXPECT\_EQ(v1, v3);

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

TDynamicVector<int> v1(arr1, 3), v3(std::move(v1));

EXPECT\_NE(&v1, &v3);

}

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.at(-1));

}

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

{

TDynamicVector<int> v(10);

ASSERT\_ANY\_THROW(v.at(11));

}

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

{

TDynamicVector<int> v(10);

ASSERT\_NO\_THROW(v = v);

EXPECT\_EQ(v, v = v);

}

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

{

TDynamicVector<int> v(10), v1(10);

ASSERT\_NO\_THROW(v = v1);

EXPECT\_EQ(v1, v = v1);

}

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

{

TDynamicVector<int> v(10), v1(5);

ASSERT\_NO\_THROW(v = v1);

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

}

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

{

TDynamicVector<int> v(10), v1(5);

ASSERT\_NO\_THROW(v = v1);

EXPECT\_EQ(v1, v = v1);

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr3 = new int[3] { 1, 2, 3 };

TDynamicVector<int> v1(arr1, 3), v3(arr3, 3);

EXPECT\_EQ(v1, v3);

delete[] arr1;

delete[] arr3;

}

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

{

TDynamicVector<int> v1(4);

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

}

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

{

TDynamicVector<int> v1(4), v2(5);

EXPECT\_EQ(v1 == v2, false);

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr3 = new int[3] { 4, 5, 6 };

TDynamicVector<int> v1(arr1, 3), v3(arr3, 3);

EXPECT\_EQ(v3, v1 + 3);

delete[] arr1;

delete[] arr3;

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr3 = new int[3] { -2, -1, 0 };

TDynamicVector<int> v1(arr1, 3), v3(arr3, 3);

EXPECT\_EQ(v3, v1 - 3);

delete[] arr1;

delete[] arr3;

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr3 = new int[3] { 3, 6, 9 };

TDynamicVector<int> v1(arr1, 3), v3(arr3, 3);

EXPECT\_EQ(v3, v1 \* 3);

delete[] arr1;

delete[] arr3;

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr2 = new int[3] { 2, 3, 4 };

int\* arr3 = new int[3] { 3, 5, 7 };

TDynamicVector<int> v1(arr1, 3), v2(arr2, 3), v3(arr3, 3);

EXPECT\_EQ(v3, v1 + v2);

delete[] arr1;

delete[] arr2;

delete[] arr3;

}

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

{

TDynamicVector<int> v1(3), v2(4);

ASSERT\_ANY\_THROW(v1 + v2);

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr2 = new int[3] { 2, 3, 4 };

int\* arr3 = new int[3] { -1, -1, -1 };

TDynamicVector<int> v1(arr1, 3), v2(arr2, 3), v3(arr3, 3);

EXPECT\_EQ(v3, v1 - v2);

delete[] arr1;

delete[] arr2;

delete[] arr3;

}

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

{

TDynamicVector<int> v1(3), v2(4);

ASSERT\_ANY\_THROW(v1 - v2);

}

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

{

int\* arr1 = new int[3] { 1, 2, 3 };

int\* arr2 = new int[3] { 2, 3, 4 };

TDynamicVector<int> v1(arr1, 3), v2(arr2, 3);

EXPECT\_EQ(20, v1 \* v2);

delete[] arr1;

delete[] arr2;

}

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

{

TDynamicVector<int> v1(3), v2(4);

ASSERT\_ANY\_THROW(v1 \* v2);

}

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> m1(2);

for (size\_t i = 0; i < 2; i++)

for (size\_t j = 0; j < 2; j++)

m1[i][j] = i + j;

TDynamicMatrix<int> m2(m1);

EXPECT\_EQ(m1, m2);

}

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

{

TDynamicMatrix<int> m1(2);

for (size\_t i = 0; i < 2; i++)

for (size\_t j = 0; j < 2; j++)

m1[i][j] = i + j;

TDynamicMatrix<int> m2(std::move(m1));

EXPECT\_NE(&m1, &m2);

}

TEST(TDynamicMatrix, can\_get\_size)

{

TDynamicMatrix<int> m(5);

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

}

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

{

TDynamicMatrix<int> m(4);

ASSERT\_NO\_THROW(m[0][0] = 10);

EXPECT\_EQ(10, m.at(0, 0));

}

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

{

TDynamicMatrix<int> m(4);

ASSERT\_ANY\_THROW(m.at(-1, -4));

}

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

{

TDynamicMatrix<int> m(4);

ASSERT\_ANY\_THROW(m.at(5, 3));

}

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

{

TDynamicMatrix<int> m(4);

ASSERT\_NO\_THROW(m = m);

EXPECT\_EQ(m, m = m);

}

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

{

TDynamicMatrix<int> m(4), m1(4);

for (size\_t i = 0; i < 4; i++)

for (size\_t j = 0; j < 4; j++)

m[i][j] = i;

ASSERT\_NO\_THROW(m1 = m);

EXPECT\_EQ(m, m1);

}

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

{

TDynamicMatrix<int> m(2), m1(4);

m = m1;

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

}

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

{

TDynamicMatrix<int> m(4), m1(2);

for (size\_t i = 0; i < 4; i++)

for (size\_t j = 0; j < 4; j++)

m[i][j] = i;

ASSERT\_NO\_THROW(m1 = m);

EXPECT\_EQ(m, m1);

}

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

{

TDynamicMatrix<int> m(2), m1(2);

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

}

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

{

TDynamicMatrix<int> m1(2);

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

}

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

{

TDynamicMatrix<int> m1(2), m2(4);

EXPECT\_EQ(false, m1 == m2);

}

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

{

TDynamicMatrix<int> m(4), m1(4), m2(4);

for (size\_t i = 0; i < 4; i++)

for (size\_t j = 0; j < 4; j++) {

m[i][j] = i;

m1[i][j] = j;

m2[i][j] = i + j;

}

EXPECT\_EQ(m2, m + m1);

}

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

{

TDynamicMatrix<int> m(4), m1(2);

ASSERT\_ANY\_THROW(m + m1);

}

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

{

TDynamicMatrix<int> m(4), m1(4), m2(4);

for (size\_t i = 0; i < 4; i++) {

for (size\_t j = 0; j < 4; j++) {

m[i][j] = i;

m1[i][j] = j;

m2[i][j] = i - j;

}

}

EXPECT\_EQ(m2, m - m1);

}

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

{

TDynamicMatrix<int> m(4), m1(2);

ASSERT\_ANY\_THROW(m - m1);

}

TEST(TDynamicMatrix, can\_divide\_matrixes\_with\_equal\_size)

{

TDynamicMatrix<double> m(2), m1(2), m2(2);

m[0][0] = 13;

m[0][1] = 26;

m[1][0] = 39;

m[1][1] = 13;

m1[0][0] = 7;

m1[0][1] = 4;

m1[1][0] = 2;

m1[1][1] = 3;

m2[0][0] = -1;

m2[0][1] = 10;

m2[1][0] = 7;

m2[1][1] = -5;

EXPECT\_EQ(m2, m / m1);

}

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

{

TDynamicMatrix<int> m(4), m1(2);

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;

size\_t size;

int op;

cout << "Введите размер матрицы A: ";

cin >> size;

TDynamicMatrix<double> a(size);

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

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

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

{

cin >> a[i][j];

}

cout << "Введите размер матрицы B: ";

cin >> size;

TDynamicMatrix<double> b(size);

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

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

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

{

cin >> b[i][j];

}

cout << endl << "1. Сложить\n2. Вычесть\n3. Умножить\n4. Разделить\n";

cout << "Введите номер операции: ";

cin >> op;

switch (op)

{

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;

case 4:

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

break;

default:

break;

}

}

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

1. Приложение 5

experiments.cpp

#include <iostream>

#include <chrono>

#include <random>

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

enum operations {

PLUS,

MINUS,

MULTIPLY,

DIVIDE,

COPY,

COMPARE

};

auto average\_test(size\_t size, operations op, size\_t iterations = 10) {

int max\_random = 10000;

int min\_random = 0;

long long average\_time = 0;

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

TDynamicMatrix<double> matrix(size), matrix1(size);

for (size\_t i1 = 0; i1 < size; i1++) {

for (size\_t j1 = 0; j1 < size; j1++) {

matrix[i1][j1] = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

matrix1[i1][j1] = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

}

}

std::chrono::steady\_clock::time\_point begin, end;

switch (op)

{

case PLUS:

begin = std::chrono::steady\_clock::now();

matrix + matrix1;

end = std::chrono::steady\_clock::now();

break;

case MINUS:

begin = std::chrono::steady\_clock::now();

matrix - matrix1;

end = std::chrono::steady\_clock::now();

break;

case COPY:

begin = std::chrono::steady\_clock::now();

matrix = matrix1;

end = std::chrono::steady\_clock::now();

break;

case COMPARE:

begin = std::chrono::steady\_clock::now();

matrix == matrix1;

end = std::chrono::steady\_clock::now();

break;

case MULTIPLY:

begin = std::chrono::steady\_clock::now();

matrix \* matrix1;

end = std::chrono::steady\_clock::now();

break;

case DIVIDE:

begin = std::chrono::steady\_clock::now();

matrix / matrix1;

end = std::chrono::steady\_clock::now();

break;

default:

break;

}

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

average\_time += elapsed\_ms.count();

cout << i + 1 << '\t' << elapsed\_ms.count() << endl;

}

cout << "Average time: " << average\_time / static\_cast<double>(iterations) << endl;

return average\_time / static\_cast<double>(iterations);

}

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

{

auto begin = std::chrono::steady\_clock::now();

average\_test(5, DIVIDE);

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

cout << "Execution time: " << elapsed\_ms.count();

}