Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт Информационных технологий, Математики и Механики

Отчёт по лабораторной работе

Структуры хранения матриц специального вида

Выполнил:

студент ф-та ИТММ гр. 381808-2

Осипов Николай

Проверил:

ассистент каф. МО ЭВМ, ВМК

Панов А

Нижний Новгород

2019 г.

Содержание

Введение........................................................................................................................................3

Постановка задачи........................................................................................................................4

Руководство пользователя...........................................................................................................5

Руководство программиста.........................................................................................................6

Описание структур данных.........................................................................................................6

Заключение...................................................................................................................................10 Литература....................................................................................................................................11 Приложения.................................................................................................................................12

Исходный код программы..........................................................................................................15

# Введение

Матричные обозначения широко распространены в современной математике и её приложениях. Матрица – полезный аппарат для исследования многих задач теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений.

Помимо матриц общего вида, для которых наиболее естественной и наиболее часто используемой представляется программная реализация в виде двумерного массива, в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные, …). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры, и потому более эффективных. Изучению некоторых из них посвящена данная работа.

# Постановка задачи

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

* сложение/вычитание;
* копирование;
* сравнение.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

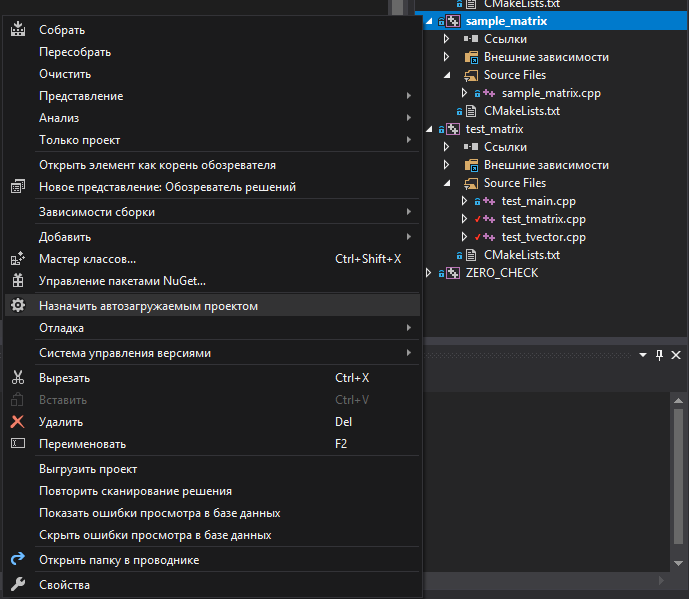
Выполнение работы предполагает решение следующих задач:

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.

**Руководство пользователя**

Пользователю предоставляется возможность воспользоваться двумя раннее написанными классам TVector и TMatrix, оформленные в файле utmatrix.h.

Пользователю необходимо лишь запустить проект sample\_matrix (ctrl+F5), назначив его автозагружаемым проектом.



Результаты можно посмотреть в Приложение 2

# Руководство программиста

## Описание структуры программы

* docs — инструкции по выполнению лабораторной работы, полезные документы.
* gtest — библиотека Google Test.
* include — директория для размещения заголовочных файлов.
* samples — директория для размещения тестового приложения.
* sln — директория с файлами решений и проектов для VS 2008 и VS 2010, вложенные директории vc9 и vc10 соответственно.
* src — директория для размещения исходных кодов (cpp-файлы).
* test — директория с модульными тестами и основным приложением, инициализирующим запуск тестов.
* README.md — информация о проекте, которую вы сейчас читаете.
* Служебные файлы
  + .gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
  + CMakeLists.txt — корневой файл для сборки проекта с помощью CMake. Может быть использован для генерации проекта в среде разработки, отличной от Microsoft Visual Studio.
  + .travis.yml — конфигурационный файл для системы автоматического тестирования Travis-CI. Тесты, входящие в состав шаблонного проекта, регулярно запускаются на удаленной инфраструктуре.

В решении содержатся следующие модули:

* Модуль utmatirx, содержащий реализацию классов Вектор и Матрица (файл ./include/utmatrix.h). Поскольку оба класса шаблонные, реализацию методов необходимо выполнять непосредственно в заголовочном файле. При этом интерфейсы классов должны оставаться неизменными.
* Тесты для классов Вектор и Матрица (файлы ./test/test\_tvector.cpp, ./test/test\_tmatrix.cpp).
* Пример использования класса Матрица (файл ./samples/sample\_matrix.cpp).

## Описание структур данных

**TVector:**

ValType \*pVector; // память для хранения элементов вектора

int Size; // размер вектора

int StartIndex; // индекс первого элемента вектора

**TMatrix:**

**class** TMatrix : **public** TVector<TVector<ValType> > // наследование полей TVector

**Описание алгоритмов**

**Класс TVector**

TVector(int s = 10, int si = 0); // одновременно конструктор по умолчанию

// и конструктор с параметрами

TVector(const TVector& v); // конструктор копирования

~TVector();

int GetSize() { return Size; } // размер вектора

int GetStartIndex() { return StartIndex; } // индекс первого элемента

ValType& operator[](int pos); // доступ

bool operator==(const TVector& v) const; // сравнение

bool operator!=(const TVector& v) const; // сравнение

TVector& operator=(const TVector& v); // присваивание

// скалярные операции

TVector operator+(const ValType& val); // прибавить скаляр

TVector operator-(const ValType& val); // вычесть скаляр

TVector operator\*(const ValType& val); // умножить на скаляр

// векторные операции

TVector operator+(const TVector& v); // сложение

TVector operator-(const TVector& v); // вычитание

ValType operator\*(const TVector& v); // скалярное произведение

**Класс TMatrix** является вектором векторов инет смысла его описывать, так как он использует методы Tvector.

Однако можно посмотреть исходный код каждого шаблонного класса после Приложения 2

# Эксперименты

В качестве эксперимента были использованы готовые тесты с целью проверки классов на работоспособность. За данную процедуру отвечает проект "test\_matrix", который вмещает в себя 3 файла:

test\_main.cpp основной файл в проекте, который отвечает за запуск тестов для двух классов.

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

{

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

return RUN\_ALL\_TESTS();

}

Файл test\_tvector.cpp содержит в себе список тестов, необходимых для проверки класса TVector с целью обнаружения ошибок в самом классе (см. Приложение 1)

Файл test\_tmatrix.cpp содержит в себе список тестов, необходимых для проверки класса TMatrix с целью обнаружения ошибок в самом классе (см. Приложение 1)

# Также проект “sample\_matrix” был использован в качестве проверки, чтобы пользователь лично мог убедиться в корректной работе классов (см. В руководстве пользователя и Приложение 2)

# Заключение

В лабораторной работе были реализованы два шаблонных класса (TVector и TMatrix).

Каждый класс успешно прошел проверку через Google Test и проект sample\_matrix (см. Приложение 1 и Приложение 2)

# Литература

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.

# Приложения

## Приложение 1

## 

## 

## 

## Приложение 2

## 

## Исходный код программы

## Utmatrix.h

template <class ValType>

class TVector

{

protected:

ValType\* pVector;

int Size; // размер вектора

int StartIndex; // индекс первого элемента вектора

public:

TVector(int s = 10, int si = 0); // одновременно конструктор по умолчанию

// и конструктор с параметрами

TVector(const TVector& v); // конструктор копирования

~TVector();

int GetSize() { return Size; } // размер вектора

int GetStartIndex() { return StartIndex; } // индекс первого элемента

ValType& operator[](int pos); // доступ

bool operator==(const TVector& v) const; // сравнение

bool operator!=(const TVector& v) const; // сравнение

TVector& operator=(const TVector& v); // присваивание

// скалярные операции

TVector operator+(const ValType& val); // прибавить скаляр

TVector operator-(const ValType& val); // вычесть скаляр

TVector operator\*(const ValType& val); // умножить на скаляр

// векторные операции

TVector operator+(const TVector& v); // сложение

TVector operator-(const TVector& v); // вычитание

ValType operator\*(const TVector& v); // скалярное произведение

// ввод-вывод

friend istream& operator>>(istream& in, TVector& v)

{

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

in >> v.pVector[i];

return in;

}

friend ostream& operator<<(ostream& out, const TVector& v)

{

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

out << v.pVector[i] << ' ';

return out;

}

};

template <class ValType>

TVector<ValType>::TVector(int s, int si) :Size(s), StartIndex(si)

{

if (s > MAX\_VECTOR\_SIZE || s < 0 || si < 0)

throw exception("error");

pVector = new ValType[Size];

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

pVector[i] = (ValType)0;

}

template <class ValType> //конструктор копирования

TVector<ValType>::TVector(const TVector<ValType>& v)

{

Size = v.Size;

StartIndex = v.StartIndex;

pVector = new ValType[Size];

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

pVector[i] = v.pVector[i];

}

template <class ValType>

TVector<ValType>::~TVector()

{

if (Size)

delete[] pVector;

Size = 0;

StartIndex = 0;

pVector = NULL;

}

template <class ValType> // доступ

ValType& TVector<ValType>::operator[](int pos)

{

if (pos < 0 || pos >= StartIndex + Size)

throw exception("error");

return pVector[pos - StartIndex];

}

template <class ValType> // сравнение

bool TVector<ValType>::operator==(const TVector& v) const

{

if (Size != v.Size)

return false;

if (StartIndex != v.StartIndex)

return false;

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

if (pVector[i] != v.pVector[i])

return false;

return true;

}

template <class ValType> // сравнение

bool TVector<ValType>::operator!=(const TVector& v) const

{

if (Size != v.Size)

return true;

if (StartIndex != v.StartIndex)

return true;

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

if (pVector[i] != v.pVector[i])

return true;

return false;

}

template <class ValType> // присваивание

TVector<ValType>& TVector<ValType>::operator=(const TVector& v)

{

if (this != &v)

{

if (Size != v.Size)

{

delete[] pVector;

pVector = new ValType[v.Size];

Size = v.Size;

}

StartIndex = v.StartIndex;

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

pVector[i] = v.pVector[i];

}

return \*this;

}

template <class ValType> // прибавить скаляр

TVector<ValType> TVector<ValType>::operator+(const ValType& val)

{

TVector<ValType> tmp(\*this);

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

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

return tmp;

}

template <class ValType> // вычесть скаляр

TVector<ValType> TVector<ValType>::operator-(const ValType& val)

{

TVector<ValType> tmp(\*this);

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

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

return tmp;

}

template <class ValType> // умножить на скаляр

TVector<ValType> TVector<ValType>::operator\*(const ValType& val)

{

TVector<ValType> tmp(\*this);

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

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

return tmp;

}

template <class ValType> // сложение

TVector<ValType> TVector<ValType>::operator+(const TVector<ValType>& v)

{

if (Size != v.Size)

throw "Error";

TVector<ValType> tmp(\*this);

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

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

return tmp;

}

template <class ValType> // вычитание

TVector<ValType> TVector<ValType>::operator-(const TVector<ValType>& v)

{

if (Size != v.Size)

throw "Error";

TVector<ValType> tmp(\*this);

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

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

return tmp;

}

template <class ValType> // скалярное произведение

ValType TVector<ValType>::operator\*(const TVector<ValType>& v)

{

if (Size != v.Size)

throw "Error";

ValType tmp = 0;

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

tmp += pVector[i] \* v.pVector[i];

return tmp;

}

// Верхнетреугольная матрица

template <class ValType>

class TMatrix : public TVector<TVector<ValType> >

{

public:

TMatrix(int s = 10);

TMatrix(const TMatrix& mt); // копирование

TMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа

bool operator==(const TMatrix& mt) const; // сравнение

bool operator!=(const TMatrix& mt) const; // сравнение

TMatrix& operator= (const TMatrix& mt); // присваивание

TMatrix operator+ (const TMatrix& mt); // сложение

TMatrix operator- (const TMatrix& mt); // вычитание

TMatrix operator\* (const TMatrix& mt); // умножение

// ввод / вывод

friend istream& operator>>(istream& in, TMatrix& mt)

{

for (int i = 0; i < mt.Size; i++)

in >> mt.pVector[i];

return in;

}

friend ostream& operator<<(ostream& out, const TMatrix& mt)

{

for (int i = 0; i < mt.Size; i++)

out << mt.pVector[i] << endl;

return out;

}

};

template <class ValType>

TMatrix<ValType>::TMatrix(int s) : TVector<TVector<ValType> >(s)

{

if (s > MAX\_MATRIX\_SIZE)

throw "Error";

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

{

TVector<ValType> t(s - i, i);

pVector[i] = t;

}

// по умолчанию создается квадратная матрица sхs

// надо заменить созданную матрицу верхнетреугольной

// вектора должны быть разной длины в матрице

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // конструктор копирования

TMatrix<ValType>::TMatrix(const TMatrix<ValType>& mt) :

TVector<TVector<ValType> >(mt) {}

template <class ValType> // конструктор преобразования типа

TMatrix<ValType>::TMatrix(const TVector<TVector<ValType> >& mt) :

TVector<TVector<ValType> >(mt) {}

template <class ValType> // сравнение

bool TMatrix<ValType>::operator==(const TMatrix<ValType>& mt) const

{

if (Size != mt.Size)

return false;

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

if (pVector[i] != mt.pVector[i])

return false;

return true;

}

template <class ValType> // сравнение

bool TMatrix<ValType>::operator!=(const TMatrix<ValType>& mt) const

{

if (Size != mt.Size)

return true;

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

if (pVector[i] != mt.pVector[i])

return true;

return false

}

template <class ValType> // присваивание

TMatrix<ValType>& TMatrix<ValType>::operator=(const TMatrix<ValType>& mt)

{

if (this != &mt)

{

if (Size != mt.Size)

{

Size = mt.Size;

delete[] pVector;

pVector = new TVector<ValType>[mt.Size];

}

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

pVector[i] = mt.pVector[i];

}

return \*this;

}

template <class ValType> // сложение

TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType>& mt)

{

TMatrix<ValType> tmp(\*this);

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

tmp.pVector[i] = pVector[i] + mt.pVector[i];

return tmp;

}

template <class ValType> // вычитание

TMatrix<ValType> TMatrix<ValType>::operator-(const TMatrix<ValType>& mt)

{

TMatrix<ValType> tmp(\*this);

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

tmp.pVector[i] = pVector[i] - mt.pVector[i];

return tmp;

}

template<class ValType>

TMatrix<ValType> TMatrix<ValType>::operator\*(const TMatrix<ValType>& mt) // умножение

{

if (Size != mt.Size)

throw "error";

TMatrix<ValType> tmp(Size);

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

for (int j = i; j < Size; j++) {

for (int k = i; k <= j; k++) {

tmp.pVector[i][j] += pVector[i][k] \* mt.pVector[k][j];

}

}

}

return tmp;

}

// TVector О3 Л2 П4 С6

// TMatrix О2 Л2 П3 С3

## #endif

## Test\_tvector.cpp

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

{

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

}

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

{

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

}

TEST(TVector, can\_create\_vector\_with\_max\_size)

{

ASSERT\_NO\_THROW(TVector<int> v(MAX\_VECTOR\_SIZE));

}

TEST(TVector, can\_create\_vector\_with\_zero\_size)

{

ASSERT\_NO\_THROW(TVector<int> v(0));

}

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

{

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

}

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_startindex)

{

ASSERT\_ANY\_THROW(TVector<int> v(5, -2));

}

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

{

TVector<int> v(10);

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

}

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

{

TVector<int> v1(10);

v1[0] = 1;

v1[5] = 2;

TVector<int> v2(v1);

ASSERT\_EQ(v1, v2);

}

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

{

TVector<int> v1(10);

TVector<int> v2(v1);

ASSERT\_NE(&v1, &v2);

}

TEST(TVector, can\_get\_size)

{

TVector<int> v(4);

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

}

TEST(TVector, can\_get\_start\_index)

{

TVector<int> v(4, 2);

EXPECT\_EQ(2, v.GetStartIndex());

}

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

{

TVector<int> v(4);

v[0] = 4;

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

}

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

{

TVector<int> v(5);

ASSERT\_ANY\_THROW(v[-1] = 2);

}

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

{

TVector<int> v(5);

ASSERT\_ANY\_THROW(v[5] = 2);

}

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

{

TVector<int> v(5);

ASSERT\_NO\_THROW(v = v);

}

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

{

TVector<int> v1(4);

TVector<int> v2(4);

ASSERT\_NO\_THROW(v1 = v2);

}

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

{

TVector<int> v1(5);

TVector<int> v2(3);

v2 = v1;

EXPECT\_EQ(5, v2.GetSize());

}

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

{

TVector<int> v1(5);

TVector<int> v2(3);

ASSERT\_NO\_THROW(v2 = v1;);

}

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

{

TVector<int> v1(5);

v1[2] = 3;

TVector<int> v2(v1);

ASSERT\_EQ(v1, v2);

}

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

{

TVector<int> v1(5);

ASSERT\_EQ(v1, v1);

}

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

{

TVector<int> v1(5);

TVector<int> v2(3);

ASSERT\_NE(v1, v2);

}

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

{

TVector<int> v1(5);

ASSERT\_NO\_THROW(v1 + 10);

}

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

{

TVector<int> v1(5);

ASSERT\_NO\_THROW(v1 - 10);

}

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

{

TVector<int> v1(5);

ASSERT\_NO\_THROW(v1 \* 10);

}

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

{

TVector<int> v1(5);

TVector<int> v2(5);

ASSERT\_NO\_THROW(v1 + v2);

}

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

{

TVector<int> v1(5);

TVector<int> v2(3);

ASSERT\_ANY\_THROW(v1 + v2);

}

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

{

TVector<int> v1(5);

TVector<int> v2(5);

ASSERT\_NO\_THROW(v1 - v2);

}

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

{

TVector<int> v1(5);

TVector<int> v2(3);

ASSERT\_ANY\_THROW(v1 - v2);

}

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

{

TVector<int> v1(4);

v1[0] = 2;

v1[1] = 1;

v1[2] = 1;

v1[3] = 1;

TVector<int> v2(4);

v2[0] = 2;

v2[1] = 1;

v2[2] = 1;

v2[3] = 1;

EXPECT\_EQ(7, v1 \* v2);

}

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

{

TVector<int> v1(2);

v1[0] = 1;

v1[1] = 2;

TVector<int> v2(3);

v2[0] = 1;

v2[1] = 3;

v2[2] = 9;

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

## }

## Test\_tmatrix.cpp

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

{

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

}

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

{

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

}

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

{

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

}

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

{

TMatrix<int> m(5);

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

}

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

{

TMatrix<int> m1(5);

TMatrix<int> m2(m1);

ASSERT\_EQ(m1, m2);

}

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

{

TMatrix<int> m1(5);

TMatrix<int> m2(m1);

ASSERT\_NE(&m1, &m2);

}

TEST(TMatrix, can\_get\_size)

{

int size = 5;

ASSERT\_NO\_THROW(TMatrix<int> m(size));

}

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

{

TMatrix<int> m(5);

m[1][1] = 4;

EXPECT\_EQ(4, m[1][1]);

}

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

{

TMatrix<int> m(5);

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

}

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

{

TMatrix<int> m(5);

ASSERT\_ANY\_THROW(m[15][13] = 5);

}

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

{

TMatrix<int> m(5);

m[0][1] = 3;

m[1][2] = 9;

m[3][3] = 5;

ASSERT\_NO\_THROW(m = m);

}

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

{

TMatrix<int> m1(2);

TMatrix<int> m2(2);

m1[0][0] = 1;

m1[0][1] = 2;

m1[1][1] = 4;

m2 = m1;

EXPECT\_EQ(1, m2 == m1);

}

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

{

TMatrix<int> m1(5);

TMatrix<int> m2(7);

m2 = m1;

EXPECT\_EQ(5, m2.GetSize());

}

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

{

TMatrix<int> m1(5);

TMatrix<int> m2(10);

m1[0][0] = 1;

m1[0][1] = 2;

m1[1][1] = 4;

m2 = m1;

EXPECT\_EQ(1, m2 == m1);

}

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

{

TMatrix<int> m1(2);

TMatrix<int> m2(2);

m1[0][0] = 1; m2[0][0] = 1;

m1[0][1] = 1; m2[0][1] = 1;

m1[1][1] = 1; m2[1][1] = 1;

EXPECT\_EQ(1, m1 == m2);

}

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

{

TMatrix<int> m(2);

m[0][0] = 1;

m[0][1] = 2;

m[1][1] = 3;

EXPECT\_EQ(1, m == m);

}

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

{

TMatrix<int> m1(2);

TMatrix<int> m2(3);

EXPECT\_NE(1, m1 == m2);

}

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

{

TMatrix<int> m1(2);

TMatrix<int> m2(2);

TMatrix<int> m3(2);

m1[0][0] = 1;

m1[0][1] = 1;

m1[1][1] = 1;

m2[0][0] = 1;

m2[0][1] = 1;

m2[1][1] = 1;

m3[0][0] = 2;

m3[0][1] = 2;

m3[1][1] = 2;

EXPECT\_EQ(m3, m1 + m2);

}

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

{

TMatrix<int> m1(4);

TMatrix<int> m2(3);

ASSERT\_ANY\_THROW(m1 + m2);

}

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

{

TMatrix<int> m1(2);

TMatrix<int> m2(2);

TMatrix<int> m3(2);

m1[0][0] = 1;

m1[0][1] = 1;

m1[1][1] = 1;

m2[0][0] = 1;

m2[0][1] = 1;

m2[1][1] = 1;

m3[0][0] = 0;

m3[0][1] = 0;

m3[1][1] = 0;

EXPECT\_EQ(m3, m1 - m2);

}

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

{

TMatrix<int> m1(4);

TMatrix<int> m2(3);

ASSERT\_ANY\_THROW(m1 - m2);

}

TEST(TMatrix, can\_multiply\_matrices\_with\_equal\_size)

{

TMatrix<int> m1(3);

TMatrix<int> m2(3);

TMatrix<int> m3(3);

m1[0][0] = 1;

m1[0][1] = 1;

m1[0][2] = 1;

m1[1][1] = 1;

m1[1][2] = 1;

m1[2][2] = 1;

m2[0][0] = 1;

m2[0][1] = 1;

m2[0][2] = 1;

m2[1][1] = 1;

m2[1][2] = 1;

m2[2][2] = 1;

m3[0][0] = 1;

m3[0][1] = 2;

m3[0][2] = 3;

m3[1][1] = 1;

m3[1][2] = 2;

m3[2][2] = 1;

EXPECT\_EQ(m3, m1 \* m2);

}

TEST(TMatrix, can\_multiply\_matrices\_with\_not\_equal\_size)

{

TMatrix<int> m1(6);

TMatrix<int> m2(2);

ASSERT\_ANY\_THROW(m1 \* m2);

## }