Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Разработка структуры данных для хранения верхнетреугольных матриц**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

Матрица – это важный математический объект, применяющийся для решения большого количества задач. Во многих из них используются симметричные относительно главной диагонали матрицы, в этом случае хранить все элементы не обязательно, ведь вторую половину матрицы однозначно можно восстановить по первой.

**Постановка задачи**

Реализовать структуру данных “верхнетреугольная матрица” как шаблонный класс, написать тесты, проверяющие корректность его работы, и добиться

правильного выполнения тестовой программы, производящей арифметические действия над верхнетреугольными матрицами.

**Описание структуры программы**

Для реализации класса “верхнетреугольная матрица” необходимо сначала создать и описать структуру данных “вектор”, немного модифицированную под данную задачу, и на ней базировать матрицу как наследника класса “вектор, состоящий из векторов”. Модификация заключается в следующем: у каждого вектора есть индекс первого элемента – число, с которого начинается нумерация элементов этого вектора в составе матрицы.

Программа содержит в себе 3 проекта:

1. “sample\_matrix” (реализация векторов и матриц, тестовая программа)
2. “test\_utmatrix” (тесты для векторов и матриц)
3. “gtest” (библиотека Google Test)

**sample\_matrix**

Содержит 2 файла:

1. utmatrix.h
2. sample\_matrix.cpp

**utmatrix.h –** объявление классов TVector (вектор) и TMatrix (матрица), описание их полей и методов

ValType – шаблонный класс

TVector

Поля:

protected:

ValType \*pVector; - память для представления вектора

int Size; - размер вектора

int StartIndex; - индекс первого элемента вектора

Методы:

public:

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

TVector(const TVector &v); - конструктор копирования

~TVector(); - деструктор

int GetSize() {…} – получить размер вектора (реализован при объявлении)

int GetStartIndex() {…} – получить индекс первого элемента (реализован при объявлении)

ValType& operator[](int pos); - перегрузка оператора индексации

bool operator==(const TVector &v) const; - перегрузка операции сравнения

bool operator!=(const TVector &v) const; - перегрузка операции не равно

TVector& operator=(const TVector &v); - перегрузка оператора присваивания

TVector operator+(const ValType &val); - перегрузка операции сложения (прибавить скаляр)

TVector operator-(const ValType &val); - перегрузка операции вычитания (вычесть скаляр)

TVector operator\*(const ValType &val); - перегрузка операции умножения (умножить на скаляр)

TVector operator+(const TVector &v); - перегрузка операции сложения (сложение векторов)

TVector operator-(const TVector &v); - перегрузка операции вычитания (вычитание векторов)

ValType operator\*(const TVector &v); - перегрузка операции умножения (скалярное произведение)

friend istream& operator>>(istream &in, TVector &v) {…} – ввод из потока (реализован при объявлении)

friend ostream& operator<<(ostream &out, const TVector &v) {…} – вывод в поток (реализован при объявлении)

TMatrix

Методы:

TMatrix(int s = 10); - конструктор по умолчанию

TMatrix(const TMatrix &mt); -конструктор копирования

TMatrix(const TVector<TVector<ValType> > &mt); - конструктор преобразования типа

bool operator==(const TMatrix &mt) const; - перегрузка операции сравнения

bool operator!=(const TMatrix &mt) const; - перегрузка операции не равно

TMatrix& operator= (const TMatrix &mt); - перегрузка операции присваивания

TMatrix operator+ (const TMatrix &mt); - перегрузка операции сложения (сложение матриц)

TMatrix operator- (const TMatrix &mt); - перегрузка операции вычитания (вычитание матриц)

friend istream& operator>>(istream &in, TMatrix &mt) {…} - ввод из потока (реализован при объявлении)

friend ostream & operator<<( ostream &out, const TMatrix &mt) {…} - вывод в поток (реализован при объявлении)

**sample\_matrix.cpp –** тестовая программа

В ней заводятся 3 матрицы, первые две из них заполняются, а третья является результатом сложения первой и второй. Все 3 матрицы выводятся на экран.

**test\_umatrix**

Содержит 3 файла:

1. “test\_main.cpp” (отвечает за запуск всех тестов)
2. “test\_tvector.cpp”
3. “test\_tmatrix.cpp”

**test\_tvector.cpp –** тесты для класса TVector

TEST(TVector, can\_create\_vector\_with\_positive\_length) – можно создать вектор с положительной длиной

TEST(TVector, cant\_create\_too\_large\_vector) – нельзя создать вектор слишком большой длины

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_length) – выбрасывается исключение, когда создаётся вектор с отрицательной длиной

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_startindex) – выбрасывается исключение, когда создаётся вектор с отрицательным индексом первого элемента

TEST(TVector, can\_create\_copied\_vector) – можно создать вектор-копию

TEST(TVector, copied\_vector\_is\_equal\_to\_source\_one) – скопированный вектор соответствует оригиналу

TEST(TVector, copied\_vector\_has\_its\_own\_memory) – у вектора-копии есть своя память

TEST(TVector, can\_get\_size) – можно получить размер вектора

TEST(TVector, can\_get\_start\_index) – можно получить индекс первого элемента

TEST(TVector, can\_set\_and\_get\_element) – можно вставить и получить элемент

TEST(TVector, throws\_when\_set\_element\_with\_negative\_index) ) – выбрасывается исключение, когда пытаются получить доступ к элементу с отрицательным индексом

TEST(TVector, throws\_when\_set\_element\_with\_too\_large\_index) ) – выбрасывается исключение, когда пытаются получить доступ к элементу с слишком большим индексом

TEST(TVector, can\_assign\_vector\_to\_itself) – можно присвоить вектор самому себе

TEST(TVector, can\_assign\_vectors\_of\_equal\_size) – можно присвоить векторы одинакового размера

TEST(TVector, assign\_operator\_change\_vector\_size) – оператор присваивания изменяет размеры вектора, если они не совпадают

TEST(TVector, can\_assign\_vectors\_of\_different\_size) – можно присвоить векторы разных размеров

TEST(TVector, compare\_equal\_vectors\_return\_true) – сравнение одинаковых векторов возвращает истину

TEST(TVector, compare\_vector\_with\_itself\_return\_true) – сравнение вектора с самим собой возвращает истину

TEST(TVector, vectors\_with\_different\_size\_are\_not\_equal) – векторы разных размеров не равны

TEST(TVector, can\_add\_scalar\_to\_vector) – можно прибавить скаляр к вектору

TEST(TVector, can\_subtract\_scalar\_from\_vector) – можно вычесть скаляр из вектора

TEST(TVector, can\_multiply\_scalar\_by\_vector) – можно умножить вектор на скаляр

TEST(TVector, can\_add\_vectors\_with\_equal\_size) – можно сложить векторы одинаковых размеров

TEST(TVector, cant\_add\_vectors\_with\_not\_equal\_size) – нельзя сложить векторы разных размеров

TEST(TVector, can\_subtract\_vectors\_with\_equal\_size) – можно вычесть векторы одинакового размера

TEST(TVector, cant\_subtract\_vectors\_with\_not\_equal\_size) – нельзя вычесть векторы разных размеров

TEST(TVector, can\_multiply\_vectors\_with\_equal\_size) – можно умножить векторы одинакового размера

TEST(TVector, cant\_multiply\_vectors\_with\_not\_equal\_size) – нельзя умножить векторы разных размеров

**test\_tmatrix.cpp –** тесты для класса TMatrix

TEST(TMatrix, can\_create\_matrix\_with\_positive\_length) – можно создать матрицу положительногоразмера

TEST(TMatrix, cant\_create\_too\_large\_matrix) – нельзя создать слишком большую матрицу

TEST(TMatrix, throws\_when\_create\_matrix\_with\_negative\_length) – выбрасывается исключение, когда создаётся матрица отрицательного размера

TEST(TMatrix, can\_create\_copied\_matrix) – можно создать матрицу-копию

TEST(TMatrix, copied\_matrix\_is\_equal\_to\_source\_one) – скопированная матрица соответствует оригиналу

TEST(TMatrix, copied\_matrix\_has\_its\_own\_memory) – у матрицы-копии есть своя память

TEST(TMatrix, can\_get\_size) – можно получить размер

TEST(TMatrix, can\_set\_and\_get\_element) – можно вставить и получить элемент

TEST(TMatrix, throws\_when\_set\_element\_with\_negative\_index) – выбрасывается исключение, когда пытаются получить доступ к элементу с отрицательным индексом

TEST(TMatrix, throws\_when\_set\_element\_with\_too\_large\_index) – выбрасывается исключение, когда пытаются получить доступ к элементу с слишком большим индексом

TEST(TMatrix, can\_assign\_matrix\_to\_itself) – можно присвоить матрицу самой себе

TEST(TMatrix, can\_assign\_matrices\_of\_equal\_size) – можно сравнить матрицы одинаковых размеров

TEST(TMatrix, assign\_operator\_change\_matrix\_size) – оператор присваивания меняет размер матрицы

TEST(TMatrix, can\_assign\_matrices\_of\_different\_size) – можно присвоить матрицы разных размеров

TEST(TMatrix, compare\_equal\_matrices\_return\_true) – сравнение одинаковых матриц возвращает истину

TEST(TMatrix, compare\_matrix\_with\_itself\_return\_true) – сравнение матрицы с самой собой возвращает истину

TEST(TMatrix, matrices\_with\_different\_size\_are\_not\_equal) – матрицы разного размера не равны

TEST(TMatrix, can\_add\_matrices\_with\_equal\_size) – можно сложить матрицы одинакового размера

TEST(TMatrix, cant\_add\_matrices\_with\_not\_equal\_size) - нельзя сложить матрицы разного размера

TEST(TMatrix, can\_subtract\_matrices\_with\_equal\_size) – можно вычитать матрицы одного размера

TEST(TMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size) – нельзя вычитать матрицы разного размера

Проект “gtest” был предоставлен полностью готовым для работы, поэтому его подробное описание не приводится.

**Описание алгоритмов**

**Класс TVector**

TVector(int s = 10, int si = 0) {если s меньше 0, или s больше максимально допустимого значения, или si меньше 0, то выбросить исключение. Присвоить размеру s, индексу первого элемента – si, выделить память размера вектора}

TVector(const TVector &v) {присвоить размеру размер v, присвоить индексу первого элемента индекс v, выделить память размера вектора, скопировать все элементы вектора v}

~TVector() {освободить память}

int GetSize() {вернуть размер вектора}

int GetStartIndex() {вернуть индекс первого элемента}

ValType& operator[](int pos) {если pos меньше индекса первого элемента или больше суммы размера и индекса, то выбросить исключение. Вернуть элемент вектора под номером (pos - индекс)}

bool operator==(const TVector &v) const {если размеры не совпадают, то вернуть ложь. Если хотя бы один из элементов не совпадает, вернуть ложь. Вернуть истину}

bool operator!=(const TVector &v) const {Вернуть отрицание от сравнения \*this и v}

TVector& operator=(const TVector &v) {если this не равен адресу v, то если размеры не совпадают, присвоить размер v, освободить старую память, выделить новую; присвоить индекс первого элемента v, присвоить все элементы v в \*this. Вернуть \*this.}

TVector operator+(const ValType &val) {создать вектор под результат такую же, как \*this. Прибавить к каждому элементу из результата значение val. Вернуть результат}

TVector operator-(const ValType &val) {создать вектор под результат такую же, как \*this. Вычесть из каждого элемента из результата значение val. Вернуть результат}

TVector operator\*(const ValType &val) {создать вектор под результат такую же, как \*this. Умножить каждый элемент из результата на val. Вернуть результат}

TVector operator+(const TVector &v) {если размеры не совпадают, то выбросить исключение. Создать вектор под результат такого же размера и с тем же индексом, как и у исходного. В цикле от 0 до размера результата: в i-ый элемент результата записать сумму i-ых элементов \*this и v. Вернуть результат}

TVector operator-(const TVector &v) {если размеры не совпадают, то выбросить исключение. Создать вектор под результат такого же размера и с тем же индексом, как и у исходного. В цикле от 0 до размера результата: в i-ый элемент результата записать разность i-ых элементов \*this и v. Вернуть результат}

ValType operator\*(const TVector &v) {если размеры не совпадают, то выбросить исключение. Создать результат типа ValType и приравнять его к 0. В цикле от 0 до размера: прибавить к результату произведение i-ых элементов \*this и v. Вернуть результат}

friend istream& operator>>(istream &in, TVector &v) {в цикле от 0 до размера v: вводить из потока i-ый элемент вектора. Вернуть поток}

friend ostream& operator<<(ostream &out, const TVector &v) { в цикле от 0 до размера v: выводить в поток i-ый элемент вектора. Вернуть поток}

**Класс TMatrix**

TMatrix(int s = 10) {если s меньше 0 или больше максимально допустимого значения, то выбросить исключение. В цикле от 0 до размера матрицы: создать временный вектор с параметрами размера Size-i и индексом первого элемента i и присвоить его i-ому вектору матрицы.}

TMatrix(const TMatrix &mt) {вызвать конструктор для класса TVector<TVector<ValType> > с параметром mt}

TMatrix(const TVector<TVector<ValType> > &mt) { вызвать конструктор для класса TVector<TVector<ValType> > с параметром mt }

bool operator==(const TMatrix &mt) const {если размеры не равны, вернуть ложь. Если индексы первых элементов не равны, вернуть ложь. Если хотя бы одни из векторов не совпадают, вернуть ложь. Вернуть истину}

bool operator!=(const TMatrix &mt) const {вернуть отрицание от сравнения \*this и mt}

TMatrix& operator= (const TMatrix &mt) {если this не равен адресу mt, то если размеры не совпадают, присвоить размер mt, освободить старую память, выделить новую; присвоить индекс первого элемента mt, присвоить все элементы mt в \*this. Вернуть \*this}

TMatrix operator+ (const TMatrix &mt) {если размеры не равны, выбросить исключение. Вернуть оператор сложения для класса TVector<TVector<ValType> > с параметром mt}

TMatrix operator- (const TMatrix &mt) { если размеры не равны, выбросить исключение. Вернуть оператор вычитания для класса TVector<TVector<ValType> > с параметром mt}

friend istream& operator>>(istream &in, TMatrix &mt) {в цикле от 0 до размера матрицы: вводить из потока каждый вектор матрицы. Вернуть поток}

friend ostream & operator<<( ostream &out, const TMatrix &mt) { в цикле от 0 до размера матрицы: выводить в поток каждый вектор матрицы. Вернуть поток}

**Тесты для TVector**

TEST(TVector, can\_create\_vector\_with\_positive\_length) {ожидается, что не возникнет исключений: создать вектор длины 5}

TEST(TVector, cant\_create\_too\_large\_vector) {ожидается исключение: создать вектор длины максимум + 1}

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_length) {ожидается исключение: создать вектор длины -5}

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_startindex) { ожидается исключение: создать вектор длины 5 и индексом первого элемента -2}

TEST(TVector, can\_create\_copied\_vector) {создать вектор v размера 10. Ожидается, что не возникнет исключений: создать вектор, скопированный с v}

TEST(TVector, copied\_vector\_is\_equal\_to\_source\_one) {создать вектор а размера 3, создать вектор b. Заполнить вектор а единицами. Приравнять b к а. Ожидается, что каждая компонента вектора b будет единицей}

TEST(TVector, copied\_vector\_has\_its\_own\_memory) { создать вектор а размера 3, создать вектор b. Заполнить вектор а единицами. Приравнять b к а. Вторую компоненту b поменять на 2. Ожидается, что вторые компоненты векторов а и b будут не равны}

TEST(TVector, can\_get\_size) {создать вектор v длины 4. Ожидается, что v.GetSize() = 4}

TEST(TVector, can\_get\_start\_index) { создать вектор v длины 4 и индексом первого элемента 2. Ожидается, что v.GetStartIndex() = 2}

TEST(TVector, can\_set\_and\_get\_element) {создать вектор длины 4. Приравнять нулевую компоненту к 4. Ожидается, что нулевая компонента = 4}

TEST(TVector, throws\_when\_set\_element\_with\_negative\_index) ) {создать вектор длины 3. Ожидается исключение: попытаться получить доступ к (-1)-ому элементу}

TEST(TVector, throws\_when\_set\_element\_with\_too\_large\_index) ) {создать вектор длины 3. Ожидается исключение: попытаться получить доступ к элементу с номером максимум - 1}

TEST(TVector, can\_assign\_vector\_to\_itself) {создать вектор u размера 3. Ожидается, что не возникнет исключений: u присвоить u}

TEST(TVector, can\_assign\_vectors\_of\_equal\_size) {создать векторы u и v размера 3. u присвоить v. Ожидается равенство u и v}

TEST(TVector, assign\_operator\_change\_vector\_size) {создать вектор u размера 3 и вектор v размера 5. u присвоить v. Ожидается равенство 5 и размера u}

TEST(TVector, can\_assign\_vectors\_of\_different\_size) {создать вектор u размера 3 и вектор v размера 5. u присвоить v. Ожидается равенство u и v}

TEST(TVector, compare\_equal\_vectors\_return\_true) {создать векторы u и v размера 3. Всем компонентам обоих векторов присвоить единицу. Ожидается, что сравнение u и v вернёт истину}

TEST(TVector, compare\_vector\_with\_itself\_return\_true) {создать вектор размера 3. Ожидается, что вектор будет равен самому себе}

TEST(TVector, vectors\_with\_different\_size\_are\_not\_equal) {создать вектор u размера 3 и вектор v размера 5. Ожидается, что u и v не равны}

TEST(TVector, can\_add\_scalar\_to\_vector) {создать вектор v размера 3, состоящий из единиц, и вектор под результат размера 3. Создать целочисленную переменную а = 1. Результату присвоить сумму v и а. Ожидается, что все компоненты результата будут равны 2}

TEST(TVector, can\_subtract\_scalar\_from\_vector) {создать вектор v размера 3, состоящий из единиц, и вектор под результат размера 3. Создать целочисленную переменную а = 1. Результату присвоить разность v и а. Ожидается, что все компоненты результата будут равны 0}

TEST(TVector, can\_multiply\_scalar\_by\_vector) {создать вектор v размера 3, состоящий из единиц, и вектор под результат размера 3. Создать целочисленную переменную а = 3. Результату присвоить произведение v и а. Ожидается, что все компоненты результата будут равны 3}

TEST(TVector, can\_add\_vectors\_with\_equal\_size) {создать векторы v, u и результат размера 3. Всем компонентам v присвоить 3, u – 2. Результату присвоить сумму u и v. Ожидается, что все компоненты результата будут равны 5}

TEST(TVector, cant\_add\_vectors\_with\_not\_equal\_size) {создать вектор v размера 3, вектор u размера 5. Ожидается исключение: сложить v и u}

TEST(TVector, can\_subtract\_vectors\_with\_equal\_size) {создать векторы v, u и результат размера 3. Всем компонентам v присвоить 3, u – 2. Результату присвоить разность v и u. Ожидается, что все компоненты результата будут равны 1}

TEST(TVector, cant\_subtract\_vectors\_with\_not\_equal\_size) {создать вектор v размера 3, вектор u размера 5. Ожидается исключение: вычесть из v u}

TEST(TVector, can\_multiply\_vectors\_with\_equal\_size) {создать векторы v и u размера 3. Создать целочисленную переменную под результат. Всем компонентам v присвоить 3, u – 2. Результату присвоить произведение u и v. Ожидается, что результат будет равен 18}

TEST(TVector, cant\_multiply\_vectors\_with\_not\_equal\_size) {создать вектор v размера 3, вектор u размера 5. Ожидается исключение: умножить v и u}

**Тесты для TMatrix**

TEST(TMatrix, can\_create\_matrix\_with\_positive\_length) {ожидается, что не возникнет исключений: создать матрицу размера 5}

TEST(TMatrix, cant\_create\_too\_large\_matrix) {ожидается исключение: создать матрицу размера максимум + 1}

TEST(TMatrix, throws\_when\_create\_matrix\_with\_negative\_length) {ожидается исключение: создать матрицу размера -5}

TEST(TMatrix, can\_create\_copied\_matrix) {создать матрицу m размера 5. Ожидается, что не возникнет исключений: создать матрицу, скопированную с m}

TEST(TMatrix, copied\_matrix\_is\_equal\_to\_source\_one) {создать матрицу а размера 3, заполненную единицами, и матрицу b. b присвоить а. Ожидается, что b = а}

TEST(TMatrix, copied\_matrix\_has\_its\_own\_memory) { создать матрицу а размера 3, заполненную единицами, и матрицу b. b присвоить а. Элементу b[1][1] присвоить 2. Ожидается, что элементы b[1][1] и а[1][1] не равны}

TEST(TMatrix, can\_get\_size) {создать матрицу а размера 3. Ожидается, что а.GetSize() = 3}

TEST(TMatrix, can\_set\_and\_get\_element) {создать матрицу размера 3. Элементу [0][1] присвоить 2. Ожидается, что элемент [0][1] = 2}

TEST(TMatrix, throws\_when\_set\_element\_with\_negative\_index) {создать матрицу размера 3. Ожидается исключение: присвоить элементу [-1][-2] 2}

TEST(TMatrix, throws\_when\_set\_element\_with\_too\_large\_index) {создать матрицу размера 3. Ожидается исключение: присвоить элементу [3][3] 2}

TEST(TMatrix, can\_assign\_matrix\_to\_itself) {создать матрицу а размера 3. Ожидается выполнение без исключений: присвоить а самой себе}

TEST(TMatrix, can\_assign\_matrices\_of\_equal\_size) {создать матрицы а и b размера 3. Ожидается выполнение без исключений: присвоить а в b}

TEST(TMatrix, assign\_operator\_change\_matrix\_size) {создать матрицы а размера 3 и b размера 2. b присвоить а. Ожидается, что размер b равен 3}

TEST(TMatrix, can\_assign\_matrices\_of\_different\_size) {создать матрицы а размера 3 и b размера 2. Ожидается выполнение без исключений: присвоить b в а}

TEST(TMatrix, compare\_equal\_matrices\_return\_true) {создать матрицы а и b размера 3, составленные из единиц. Ожидается, что а и b равны}

TEST(TMatrix, compare\_matrix\_with\_itself\_return\_true) {создать матрицу размера 3. Ожидается, что а = а}

TEST(TMatrix, matrices\_with\_different\_size\_are\_not\_equal) {создать матрицы а размера 3 и b размера 4. Ожидается, что а и b не равны}

TEST(TMatrix, can\_add\_matrices\_with\_equal\_size) {создать матрицы а и b размера 3 и матрицу под результат. Результату присвоить сумму а и b. Ожидается, что результат равен сумме}

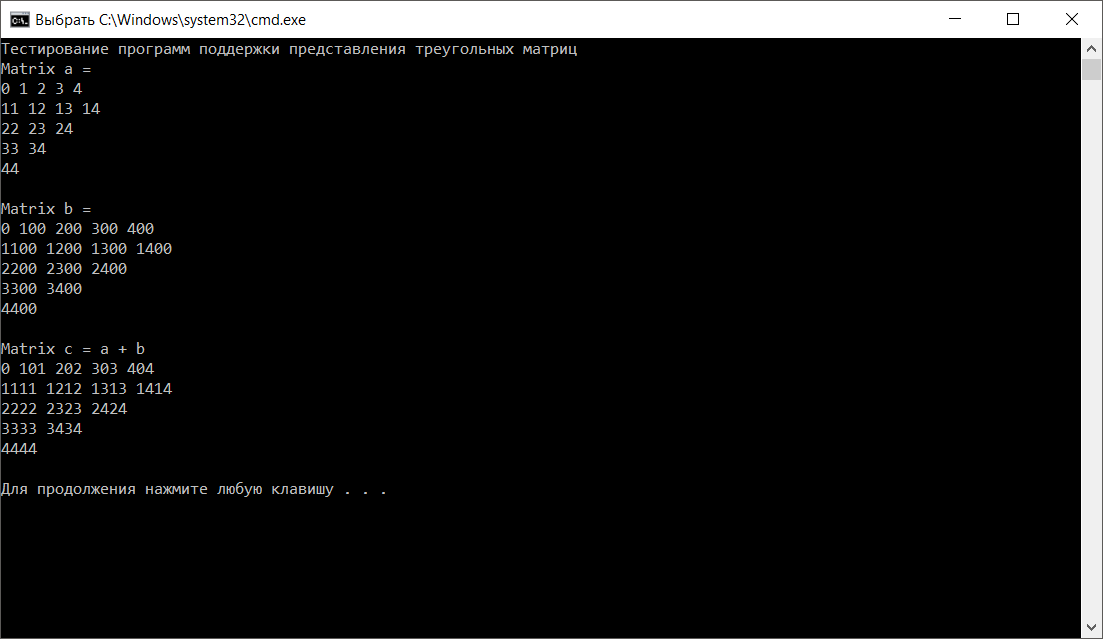
TEST(TMatrix, cant\_add\_matrices\_with\_not\_equal\_size) {создать матрицы а размера 3 и b размера 4. Ожидается исключение: сложить а и b}

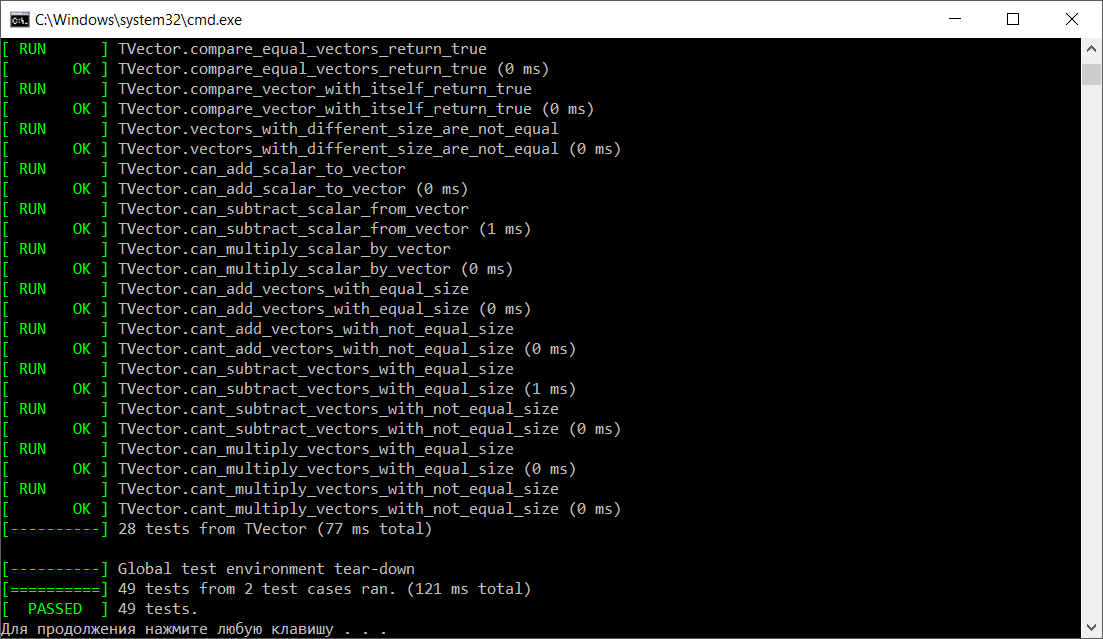
TEST(TMatrix, can\_subtract\_matrices\_with\_equal\_size) {создать матрицы а и b размера 3 и матрицу под результат. Результату присвоить разность а и b. Ожидается, что результат равен разность}

TEST(TMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size) {создать матрицы а размера 3 и b размера 4. Ожидается исключение: вычесть из а b}

**Результаты**

Как видно из скриншотов ниже, тестовая программа работает правильно, и все тесты выполняются корректно.





**Вывод**

Все тестовые программы завершаются корректно, все задачи, поставленные ранее, выполнены, а значит разработку структуры данных «верхнетреугольная матрица» можно считать успешной.

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**

**utmarix.h**

#ifndef \_\_TMATRIX\_H\_\_

#define \_\_TMATRIX\_H\_\_

#include <iostream>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000000;

const int MAX\_MATRIX\_SIZE = 10000;

// Шаблон вектора

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);

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)

{

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

throw s;

if (si < 0)

throw si;

Size = s;

StartIndex = si;

pVector = new ValType[Size];

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

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()

{

delete[] pVector;

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

template <class ValType> // доступ

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

{

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

throw pos;

return pVector[pos - StartIndex];

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

template <class ValType> // сравнение

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

{

if (Size != v.Size) 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

{

return !(\*this == v);

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

template <class ValType> // присваивание

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

{

if (this != &v) {

if (Size != v.Size) {

Size = v.Size;

delete[] pVector;

pVector = new ValType[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> res(\*this);

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

res.pVector[i] += val;

}

return res;

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

template <class ValType> // вычесть скаляр

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

{

TVector<ValType> res(\*this);

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

res.pVector[i] -= val;

}

return res;

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

template <class ValType> // умножить на скаляр

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

{

TVector<ValType> res(\*this);

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

res.pVector[i] \*= val;

}

return res;

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

template <class ValType> // сложение

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

{

if (Size != v.Size)

throw (Size - v.Size);

TVector<ValType> res(Size, StartIndex);

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

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

}

return res;

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

template <class ValType> // вычитание

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

{

if (Size != v.Size)

throw (Size - v.Size);

TVector<ValType> res(Size, StartIndex);

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

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

}

return res;

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

template <class ValType> // скалярное произведение

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

{

if (Size != v.Size)

throw (Size - v.Size);

ValType res = 0;

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

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

}

return res;

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

// 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

// 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 < 0) || (s > MAX\_MATRIX\_SIZE))

throw s;

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

TVector<ValType> tmp(Size - i, i);

pVector[i] = tmp;

}

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

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;

if (StartIndex != mt.StartIndex)

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

{

return !(\*this == mt);

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

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>[Size];

}

StartIndex = mt.StartIndex;

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)

{

if (Size != mt.Size)

throw - 1;

return TVector<TVector<ValType>>:: operator+(mt);

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

template <class ValType> // вычитание

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

{

if (Size != mt.Size)

throw - 1;

return TVector<TVector<ValType>>:: operator-(mt);

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

// TVector О3 Л2 П4 С6

// TMatrix О2 Л2 П3 С3

#endif

**test\_tvector.cpp**

#include "utmatrix.h"

#include <gtest.h>

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, 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> a(3,0),b;

for (int i = 0; i < 3; i++) {

a[i] = 1;

}

b = a;

EXPECT\_EQ(1, b[0]);

EXPECT\_EQ(1, b[1]);

EXPECT\_EQ(1, b[2]);

}

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

{

TVector<int> a(3, 0), b;

for (int i = 0; i < 3; i++) {

a[i] = 1;

}

b = a;

b[2] = 2;

EXPECT\_NE(a[2], b[2]);

}

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(3);

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

}

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

{

TVector<int> v(3);

ASSERT\_ANY\_THROW(v[MAX\_VECTOR\_SIZE +1]);

}

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

{

TVector<int> u(3);

ASSERT\_NO\_THROW(u = u);

}

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

{

TVector<int> u(3), v(3);

u = v;

EXPECT\_EQ(u, v);

}

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

{

TVector<int> u(3), v(5);

u = v;

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

}

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

{

TVector<int> u(3), v(5);

u = v;

EXPECT\_EQ(u, v);

}

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

{

TVector<int> v(3),u(3);

for (int i = 0; i < 3; i++) {

u[i] = 1;

v[i] = 1;

}

EXPECT\_EQ(v, u);

}

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

{

TVector<int> v(3);

EXPECT\_EQ(v, v);

}

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

{

TVector<int> v(3), u(5);

EXPECT\_NE(u, v);

}

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

{

TVector<int> v(3),res(3);

for (int i = 0; i < 3; i++) {

v[i] = 1;

}

int a = 1;

res = v + a;

EXPECT\_EQ(2, res[0]);

EXPECT\_EQ(2, res[1]);

EXPECT\_EQ(2, res[2]);

}

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

{

TVector<int> v(3), res(3);

for (int i = 0; i < 3; i++) {

v[i] = 1;

}

int a = 1;

res = v - a;

EXPECT\_EQ(0, res[0]);

EXPECT\_EQ(0, res[1]);

EXPECT\_EQ(0, res[2]);

}

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

{

TVector<int> v(3), res(3);

for (int i = 0; i < 3; i++) {

v[i] = 1;

}

int a = 3;

res = v \* a;

EXPECT\_EQ(3, res[0]);

EXPECT\_EQ(3, res[1]);

EXPECT\_EQ(3, res[2]);

}

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

{

TVector<int> v(3), u(3),res(3);

for (int i = 0; i < 3; i++) {

v[i] = 3;

u[i] = 2;

}

res = v + u;

EXPECT\_EQ(5, res[0]);

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

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

}

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

{

TVector<int> v(3), u(5);

ASSERT\_ANY\_THROW(v + u);

}

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

{

TVector<int> v(3), u(3), res(3);

for (int i = 0; i < 3; i++) {

v[i] = 3;

u[i] = 2;

}

res = v - u;

EXPECT\_EQ(1, res[0]);

EXPECT\_EQ(1, res[1]);

EXPECT\_EQ(1, res[2]);

}

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

{

TVector<int> v(3), u(5);

ASSERT\_ANY\_THROW(v - u);

}

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

{

TVector<int> v(3), u(3);

int res;

for (int i = 0; i < 3; i++) {

v[i] = 3;

u[i] = 2;

}

res = v \* u;

EXPECT\_EQ(18, res);

}

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

{

TVector<int> v(3), u(5);

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

}

**test\_tmatrix.cpp**

#include "utmatrix.h"

#include <gtest.h>

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> a(3), b;

for (int i = 0; i < a.GetSize(); i++) {

for (int j = i; j < a.GetSize(); j++) {

a[i][j] = 1;

}

}

b = a;

EXPECT\_EQ(a, b);

}

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

{

TMatrix<int> a(3), b;

for (int i = 0; i < a.GetSize(); i++) {

for (int j = i; j < a.GetSize(); j++) {

a[i][j] = 1;

}

}

b = a;

b[1][1] = 2;

EXPECT\_NE(a[1][1], b[1][1]);

}

TEST(TMatrix, can\_get\_size)

{

TMatrix<int> a(3);

EXPECT\_EQ(3, a.GetSize());

}

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

{

TMatrix<int> a(3);

a[0][2] = 2;

EXPECT\_EQ(2, a[0][2]);

}

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

{

TMatrix<int> a(3);

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

}

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

{

TMatrix<int> a(3);

ASSERT\_ANY\_THROW(a[3][3] = 2);

}

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

{

TMatrix<int> a(3);

ASSERT\_NO\_THROW(a = a);

}

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

{

TMatrix<int> a(3),b(3);

ASSERT\_NO\_THROW(a = b);

}

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

{

TMatrix<int> a(3), b(2);

b = a;

EXPECT\_EQ(3, b.GetSize());

}

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

{

TMatrix<int> a(3), b(2);

ASSERT\_NO\_THROW(a = b);

}

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

{

TMatrix<int> a(3), b(3);

for (int i = 0; i < a.GetSize(); i++) {

for (int j = i; j < a.GetSize(); j++) {

a[i][j] = 1;

b[i][j] = 1;

}

}

EXPECT\_EQ(a,b);

}

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

{

TMatrix<int> a(3);

EXPECT\_EQ(a, a);

}

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

{

TMatrix<int> a(3),b(4);

EXPECT\_NE(a, b);

}

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

{

TMatrix<int> a(3), b(3),c;

for (int i = 0; i < a.GetSize(); i++) {

for (int j = i; j < a.GetSize(); j++) {

a[i][j] = 3;

b[i][j] = 1;

}

}

c = a + b;

EXPECT\_EQ(c, a + b);

}

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

{

TMatrix<int> a(3), b(4);

ASSERT\_ANY\_THROW(a + b);

}

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

{

TMatrix<int> a(3), b(3), c;

for (int i = 0; i < a.GetSize(); i++) {

for (int j = i; j < a.GetSize(); j++) {

a[i][j] = 3;

b[i][j] = 1;

}

}

c = a - b;

EXPECT\_EQ(c, a - b);

}

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

{

TMatrix<int> a(3), b(4);

ASSERT\_ANY\_THROW(a - b);

}