Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

"Национальный исследовательский Нижегородский государственный

университет им. Н. И. Лобачевского"

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

СТРУКТУРА ХРАНЕНИЯ МАТРИЦ

Выполнил: студент группы 381808-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шульман Е. А.

Подпись

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А. А.

Подпись

Нижний Новгород

2019

Содержание

[**Введение** 3](#_Toc23539888)

[**Постановка задачи** 5](#_Toc23539889)

[**Руководство пользователя** 6](#_Toc23539890)

[**Руководство программиста** 9](#_Toc23539891)

[Описание структуры программы 9](#_Toc23539892)

[Описание структуры данных 10](#_Toc23539893)

[Описание алгоритмов 11](#_Toc23539894)

[**Заключение** 14](#_Toc23539895)

[**Литература** 15](#_Toc23539896)

[**Приложения** 16](#_Toc23539897)

[Приложение 1 16](#_Toc23539898)

[Приложение 2 17](#_Toc23539899)

[Приложение 3 18](#_Toc23539900)

[Фрагменты исходного кода программы 19](#_Toc23539901)

[Класс TVector 19](#_Toc23539902)

[Класс TMatrix 22](#_Toc23539903)

**Введение**

Термин «Матрица» имеет много значений.

Например, в математике матрицей называется система элементов, имеющая вид прямоугольной таблицы, в программировании матрица – это двумерный массив, в электронике – набор проводников, которые можно замкнуть в точках их пересечений. Покерные фишки также имеют непосредственное отношение к матрице. Фишки для покера изготавливаются из высококачественного композиционного материала, зачастую с металлической сердцевиной. В свою очередь композиционный материал или композит имеет матрицу и включенные в нее армирующие элементы (исключение составляют слоистые композиты).Матрица в фотографии – это интегральная микросхема (аналоговая или цифро-аналоговая), которая состоит из фотодиодов (светочувствительных элементов). Благодаря светочувствительной матрице происходит преобразование спроецированного на нее оптического изображения в электрический сигнал аналогового типа, а при наличии в составе матрицы АЦП, то преобразование происходит в поток цифровых данных. Матрица – основной элемент цифровых фотоаппаратов, всех современных видео- и телекамер, фотокамер, встроенных в мобильный телефон и системы видеонаблюдения.

Основное значение термин «матрица» имеет в математике.

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов матрицы задают размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Впервые матрицы упоминались ещё в древнем Китае, называясь тогда «волшебным квадратом». Основным применением матриц было решение линейных уравнений. Так же, волшебные квадраты были известны чуть позднее у арабских математиков, примерно тогда появился принцип сложения матриц. После развития теории определителей в конце 17-го века, Габриэль Крамер начал разрабатывать свою теорию в 18-ом столетии и опубликовал «правило Крамера» в 1751 году. Примерно в этом же промежутке времени появился «метод Гаусса». Теория матриц начала своё существование в середине XIX века в работах Уильяма Гамильтона и Артура Кэли. Фундаментальные результаты в теории матриц принадлежат Вейерштрассу, Жордану, Фробениусу. Термин «матрица» ввел Джеймс Сильвестр в 1850 г.

В данной работе будут рассмотрены верхнетреугольные матрицы, которые представляют собой квадратные матрицы, элементы которой ниже главной диагонали равны нулю.

**Постановка задачи**

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

* сложение/вычитание;
* умножение;
* копирование;
* сравнение.

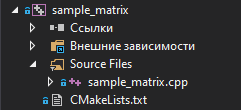
Программные средства должны содержать:

* класс Вектор (на шаблонах);
* класс Матрица (на шаблонах);

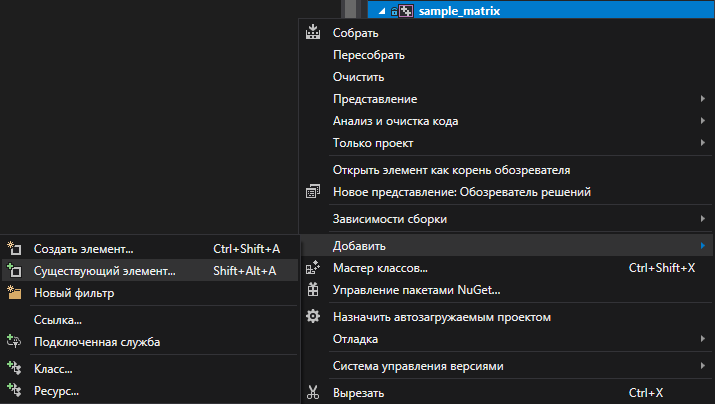
**Руководство пользователя**

По результатам работы был реализованзаголовочный файл utmatrix.h для того, чтобы его использовать необходимо:

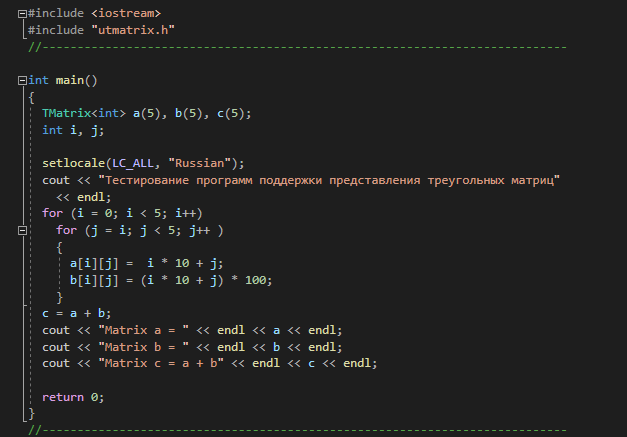
1. Создать новый проект.



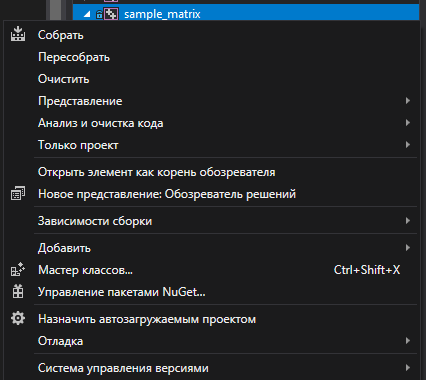
1. Добавляем существующий элемент и выбираем utmatrix.h.



1. Подключаем заголовочный файл.

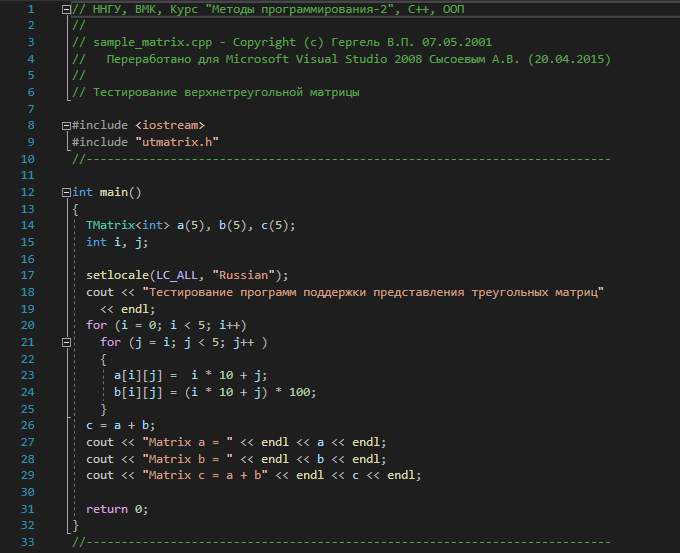


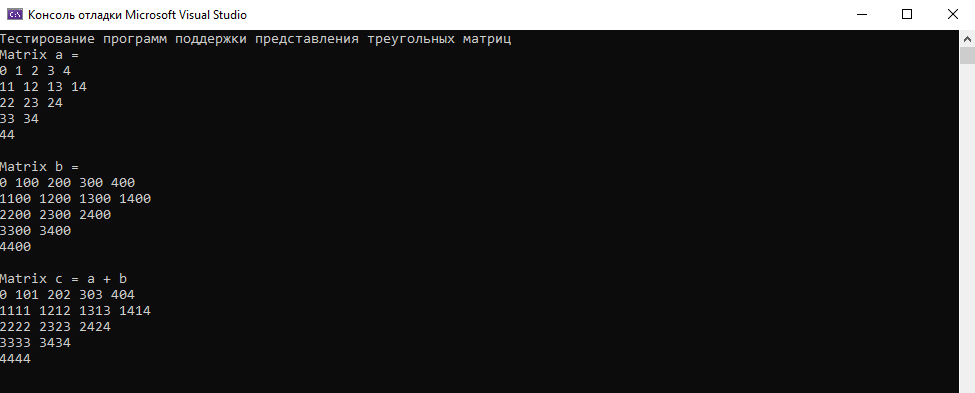
1. Назначаем наш проект автозагружаемым.



1. Теперь нашему проекту доступны функции utmatrix.h.

Проверим это:





**Руководство программиста**

Описание структуры программы

Программа состоит из 4 проектов:

* gtest– библиотека google тестов;
* sample\_matrix– тестовое приложение, позволяющее тестировать класс TMatrix;
* matrix– служебная программа, содержащая модуль utmatirx, содержащий реализацию классов TVectorи TMatrix.
* test\_matrix– тесты для классов Tvector и TMatrix;

Описание структуры данных

В данной лабораторной работе, мы храним матрицу в виде набора векторов разной длины:

Вектор

Матрица

Очевидно сходство в задании структуры данных Вектор, как набора элементов, связанных отношением следования, и структуры данных Матрица, как набора элементоввекторов, связанных отношением следования. Этот факт позволяет единообразно организовать алгоритмы обработки векторов и матриц, а, следовательно, использовать при разработке требуемых классов механизм наследования.

То есть для реализации алгоритмов нами использовано 2 класса:

* Класс «Вектор» (TVector).
* Класс «Матрицы» (TMatrix), который будет использовать класс векторов.

Описание алгоритмов

Класс TVector

Наиболее сложные методы класса

* TVector(ints = 10, int si = 0);

Конструктор создает вектор длины sи стартовым индексом si;

Если [s< 0] или [s>MAX\_VECTOR\_SIZE] или [si< 0]вызывается исключение.

Послевыделения памяти всем элементам массива pVectorприсваивается 0.

* TVector(constTVector<ValType>&v);

Конструктор копирования создает новый вектор, копируя данные из существующего v.

* int GetSize();

Данный метод возвращает размер вектора.

* int GetStartIndex();

Данный метод возвращает индекс первого элемента.

* ValType&operator[](intpos);

Перегруженный оператор [] предоставляет доступ к элементу [pos–si].

* TVector&operator=(constTVector&v);

Перегруженный оператор = копирует данные вектора vв \*this. Если [this = &v], то смысла выполнять присваивание нет - сразу возвращаем \*this. Если длины векторов различны – пересоздаем векторthis.

* booloperator==(constTVector&v) const;

Перегруженный оператор ==сравнивает левый и правый операнды и возвращает «1», если они равны, иначе возвращает«0». Если длины векторов или стартовые индексы различны – сразу возвращаем «0».

* booloperator!=(constTVector&v) const;

Перегруженный оператор !=сравнивает левый и правый операнды и возвращает «0», если они равны, иначе возвращает«1». Если длины векторов или стартовые индексы различны – сразу возвращаем «1».

* TVectoroperator+(constValType&val);

Прибавляет к каждому элементу вектора значениеval.

* TVectoroperator-(constValType&val);

Вычитает из каждого элемента вектора значениеval.

* TVectoroperator\*(constValType&val);

Умножает каждый элемент вектора на значениеval.

* TVectoroperator+(constTVector&v);

Поэлементно складывает два вектора, если длины векторов

различны - вызывается исключение.

* TVectoroperator-(constTVector&v);

Из вектора thisпоэлементновычитается вектор v, если длины векторов различны - вызывается исключение.

* TVectoroperator\*(constTVector&v);

Вычисляется скалярное произведение по формуле: ,если длины векторов различны - вызывается исключение.

* friendistream&operator>>(istream&in, TSet&v);

Перегруженный оператор >>позволяетвводить вектор.

* friendostream&operator<<(ostream&out, constTSet&v);

Перегруженный оператор <<позволяет выводить через пробел все элементы вектора.

Класс TMatrix

Наиболее сложные методы класса

* TMatrix (ints = 10);

Конструктор создает матрицу из sвекторов.

При этом, с увеличением счетчика цикла создаются векторы меньшего размера – как и было задумано.

* booloperator==(constTMatrix&mt) const;

Перегруженный оператор == сравнивает левый и правый операнды и возвращает «1», если они равны, иначе возвращает «0». Если размерматриц различны – сразу возвращаем «0».

* booloperator!=(constTMatrix&mt) const;

Перегруженный оператор != сравнивает левый и правый операнды и возвращает «0», если они равны, иначе возвращает «1». Если размерматриц различны – сразу возвращаем «1».

* TMatrix&operator= (constTMatrix&mt);

Перегруженный оператор = копирует данные матрицыmt в \*this. Если [this = &mt], то смысла выполнять присваивание нет - сразу возвращаем \*this. Если размерыматриц различны – пересоздаем матрицуthis.

* TMatrixoperator+ (constTMatrix&mt);

Построчноскладывает две матрицы.

* TMatrixoperator- (constTMatrix&mt);

Построчновычитает из матрицы thisматрицу mt.

* friendistream&operator>>(istream&in, TSet&mt);

Перегруженный оператор >>позволяетвводить матрицу.

* friendostream&operator<<(ostream&out, constTSet&mt);

Перегруженный оператор <<позволяет выводить матрицу.

**Заключение**

Целью работы была реализация структуры данных для хранения матриц специального вида (верхнетреугольных матриц).

В результате была написана структура данныхдля хранения матриц специального вида (верхнетреугольных матриц), в которой уменьшены затраты памяти для хранения векторов, путем создания векторов разной длины. Были переопределены арифметические операции для работы с матрицами, реализованными в данной структуре.

Программные средства созданы при помощи среды программирования С++ и содержат:

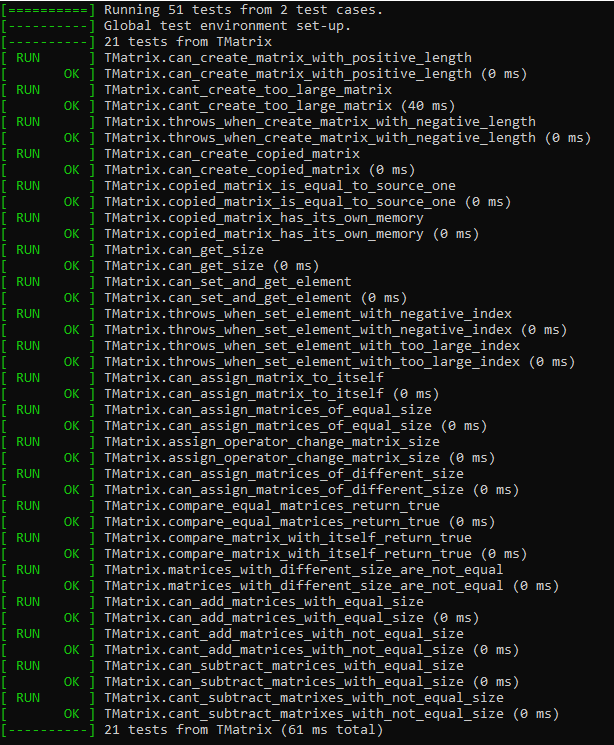
* класс Вектор (на шаблонах);
* класс Матрица (на шаблонах);

В программе также реализована система обработки ошибок.

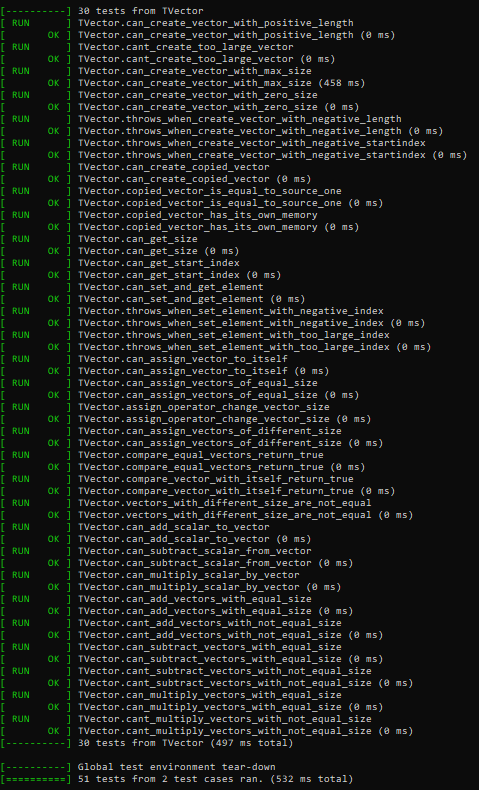
**Литература**

**Приложения**

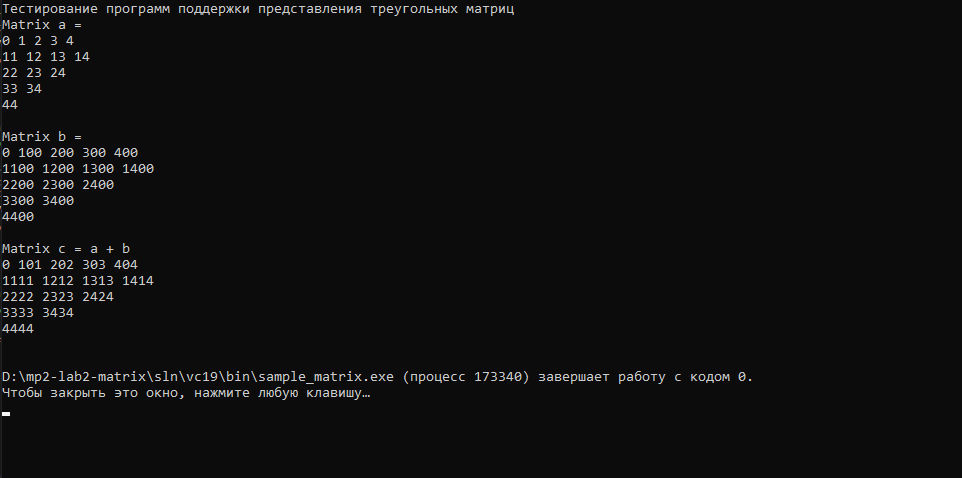
Приложение 1



Приложение 2



Приложение 3



## Фрагменты исходного кода программы

КлассTVector

// ввод-вывод

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

{

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

in>>v.pVector[i];

returnin;

}

friendostream&operator<<(ostream&out, constTVector&v)

{

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

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

returnout;

}

};

template<classValType>

TVector<ValType>::TVector(ints, intsi)

{

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

throw"Error";

if (si< 0)

throw"Error";

Size = s;

StartIndex = si;

pVector = newValType[Size];

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

pVector[i] = (ValType)0;

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

template<classValType>//конструкторкопирования

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

{

Size = v.Size;

StartIndex = v.StartIndex;

pVector = newValType[Size];

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

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

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

template<classValType>

TVector<ValType>::~TVector()

{

if (pVector != NULL)

{

delete[] pVector;

pVector = NULL;

}

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

template<classValType>// доступ

ValType&TVector<ValType>::operator[](intpos)

{

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

throw"Error";

returnpVector[pos - StartIndex];

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

template<classValType>// сравнение

boolTVector<ValType>::operator==(constTVector&v) const

{

if (Size != v.Size)

returnfalse;

if(StartIndex != v.StartIndex)

returnfalse;

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

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

returnfalse;

returntrue;

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

template<classValType>// сравнение

boolTVector<ValType>::operator!=(constTVector&v) const

{

if (Size != v.Size)

returntrue;

if (StartIndex != v.StartIndex)

returntrue;

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

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

returntrue;

returnfalse;

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

template<classValType>// присваивание

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

{

if (this != &v)

{

if (Size != v.Size)

{

delete[] pVector;

pVector = newValType[v.Size];

Size = v.Size;

}

StartIndex = v.StartIndex;

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

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

}

return \*this;

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

template<classValType>// прибавитьскаляр

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

{

TVector<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// вычестьскаляр

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

{

TVector<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// умножитьнаскаляр

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

{

TVector<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// сложение

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

{

if (Size != v.Size)

throw"Error";

if (StartIndex != v.StartIndex)

throw "Error";

TVector<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// вычитание

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

{

if (Size != v.Size)

throw"Error";

if (StartIndex != v.StartIndex)

throw "Error";

TVector<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// скалярноепроизведение

ValTypeTVector<ValType>::operator\*(constTVector<ValType>&v)

{

if (Size != v.Size)

throw"Error";

if (StartIndex != v.StartIndex)

throw "Error";

ValTypetmp = 0;

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

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

returntmp;

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

КлассTMatrix

// ввод / вывод

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

{

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

in>>mt.pVector[i];

returnin;

}

friendostream&operator<<( ostream&out, constTMatrix&mt)

{

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

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

returnout;

}

};

template<classValType>

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

{

// по умолчанию создается квадратная матрица sхs

// надо заменить созданную матрицу верхнетреугольной

// вектора должны быть разной длины в матрице

if (s> MAX\_MATRIX\_SIZE)

throw"Error";

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

{

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

pVector[i] = t;

}

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

template<classValType>// конструкторкопирования

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

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

template<classValType>// конструкторпреобразованиятипа

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

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

template<classValType>// сравнение

boolTMatrix<ValType>::operator==(constTMatrix<ValType>&mt) const

{

if(Size != mt.Size)

returnfalse;

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

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

returnfalse;

returntrue;

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

template<classValType>// сравнение

boolTMatrix<ValType>::operator!=(constTMatrix<ValType>&mt) const

{

if (Size != mt.Size)

returntrue;

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

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

returntrue;

returnfalse

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

template<classValType>// присваивание

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

{

if (this != &mt)

{

if (Size != mt.Size)

{

Size = mt.Size;

delete[] pVector;

pVector = newTVector<ValType>[mt.Size];

}

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

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

}

return \*this;

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

template<classValType>// сложение

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

{

TMatrix<ValType>tmp(\*this);

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

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

returntmp;

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

template<classValType>// вычитание

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

{

TMatrix<ValType>tmp(\*this);

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

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

returntmp;

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