## Министерство образования науки РФ

Нижегородский государственный университет имени Н.И. Лобачевского (Национальный исследовательский университет)

Институт информационных технологий математики и механики.

Отчёт по лабораторной работе №1

Тема: «Структуры хранения для матриц специального вида.»



Выполнил:

студент группы 8103

Латанов Владислав Александрович

Проверил:

Козинов Е.А.

Нижний Новгород 2015 год.

## Содержание.

1. [Введение……………………………………………………………………………3](#_Введение.)
2. [Постановка учебно-практической задачи………………………………………...4](#_Постановка_учебно-практической_зада)
3. [Руководство пользователя…………………………………………………………5](#_Руководство_пользователя.)
4. [Руководство программиста………………………………………………………..6](#_Руководство_программиста.)
5. [Заключение…………………………………………………………………………10](#_Заключение.)
6. [Список литературы………………………………………………………………...11](#_Список_используемой_литературы.)
7. [Приложение………………………………………………………………………...12](#_Приложение.)

## Введение.

**Ма́трица** — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, [целых](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), действительных 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[[1]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)#cite_note-1), в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

Для матрицы определены следующие алгебраические операции:

сложение матриц, имеющих один и тот же размер;

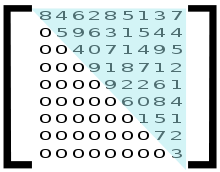
умножение матриц подходящего размера (матрицу, имеющую n столбцов, можно умножить справа на матрицу, имеющую nстрок);

в том числе умножение на матрицу вектора (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);

умножение матрицы на элемент основного кольца или поля (то есть скаляр).

**Треугольная матрица** — в линейной алгебре квадратная матрица, в которой все элементы ниже или выше главной диагонали равны нулю.

**Верхнетреугольная матрица** — квадратная матрица, в которой все элементы ниже главной диагонали равны нулю.

[](https://commons.wikimedia.org/wiki/File:Triangular_matrix.svg?uselang=ru)

**Пример верхнетреугольной матрицы**

## Постановка учебно-практической задачи.

В данной лабораторной работе, необходимо реализовать класс матриц, которые наследуются классом вектор.

* Разработанная мной программа, должна обеспечивать:

1. Сравнение векторов.
2. Присваивание векторов.
3. Скалярные операции векторов:
4. Прибавить скаляр.
5. Вычесть скаляр.
6. Умножить на скаляр.

* Векторные операции:

1. Сложение векторов.
2. Вычитание векторов.
3. Скалярное произведение векторов.

* Операции над верхнетреугольными матрицами:

1. Сравнение двух верхнетреугольных матриц.
2. Присваивание верхнетреугольных матриц.
3. Сложение верхнетреугольных матриц.
4. Вычитание верхнетреугольных матриц.

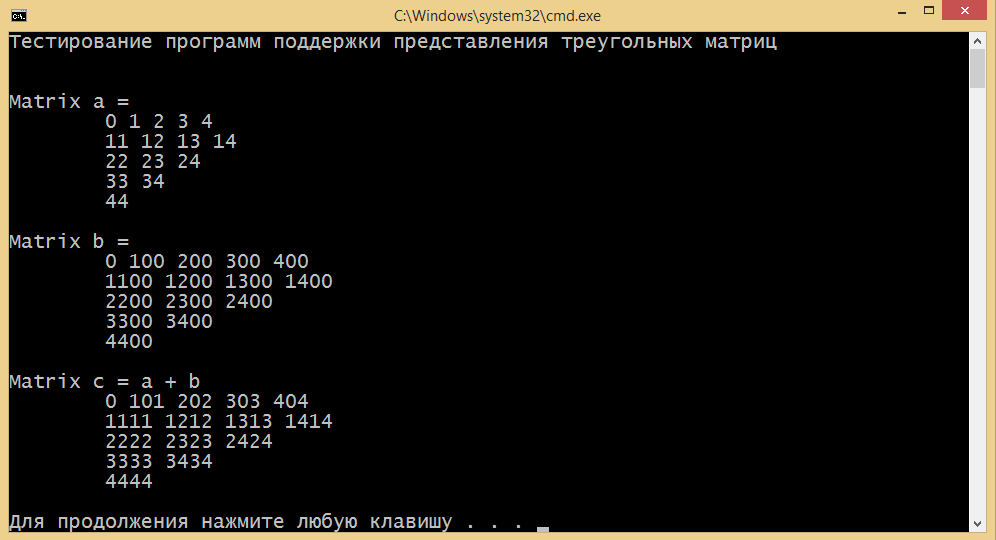
Так же моя программа должна проходить необходимый набор тестов, которые должны быть реализованы самостоятельно.

## 

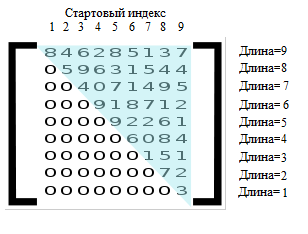
## Руководство пользователя.

В моей программе реализован пример, демонстрирующий:

1. Вывод первой матрицы.
2. Вывод второй матрицы.
3. Сумма первой и второй матриц.



## Руководство программиста.



* **Описание структур данных:**

**Матрицей с размерами *М\*М*** называется совокупность *М\*М* чисел, расположенных в виде таблицы из *М* строк и *М* столбцов. Числа, составляющие матрицу, мы будем называть элементами матрицы.

**Верхнетреугольной матрицей** называется матрица, все элементы которой ниже главной диагонали равны нулю. В данной лабораторной работе будем использовать матрицу, как вектор векторных элементов (шаблоны).

**Шаблоны** *—* средство языка C++,предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Реализовывать скалярные и векторные операции будем с помощью класса TVector, а операции над матрицами с помощью класса TMatrix. Каждый вектор имеет свою длину и начальный индекс.

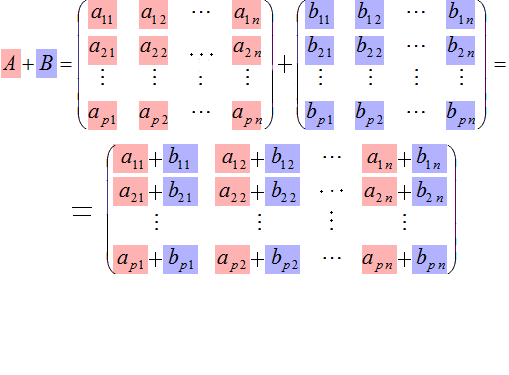
* **Описание алгоритмов:**

Действие над матрицами:

**Сложение:**

Складывать можно только матрицы одинакового размера.

Сложение матриц A + B есть операция нахождения матрицы C, все элементы которой равны попарной сумме всех соответствующих элементов матриц A и B, то есть каждый элемент матрицы C равен \ c_{ij} = a_{ij} + b_{ij}



template <class ValType> // сложение

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

{

TMatrix res(Size);

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

{

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

}

return res;

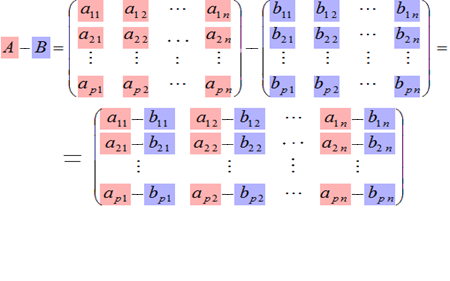
}

***Описание алгоритма:***

Создаем переменную типа TMatrix и инициализируем её размером Size. Идем по циклу и поэлементно складываем матрицы и возвращаем наш результат.

**Вычитание :**

Вычитание матриц  C:\Users\Владислав\Desktop\Безымянный.pngесть операция нахождения матрицы C, все элементы которой равны попарной сумме всех соответствующих элементов матриц A и B, то есть каждый элемент матрицы C равен C:\Users\Владислав\Desktop\Безымянный.png

****

template <class ValType> // вычитание

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

{

TMatrix res(Size);

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

{

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

}

return res;

}

***Описание алгоритма:***

Создаем переменную типа TMatrix и инициализируем её размером Size. Идем по циклу и поэлементно вычитаем матрицы и возвращаем наш результат.

**Сравнение :**

Сравнение матриц происходит почти также как и предыдущие действия, поэлементно сравниваем матрицу A и B , и если хотя бы один из элементов одной матрицы не совпадает с тем же элементом другой матрицы, то матрицы не равны, если же все элементы матриц совпадают, то матрицы , соответственно, равны.

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;

}

***Описание алгоритма:***

Для начала, мы должны проверить матрицы на то, равны ли у них размеры, ведь если у них разные размеры, то вы не можем их сравнивать и выполнять какие-либо действия над матрицами. Далее по циклу поэлементно сравниваем матрицы на неравенство, т.е. если хотя бы один элемент в двух матрицах не совпадает, то матрицы не равны, если и эта проверка пройдена успешно, то можем уверенно сказать, что матрицы равны.

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;

}

***Описание алгоритма:***

Аналогично с предыдущим, только здесь у нас цель сравнить на неравенство.

**Присваивание:**

template <class ValType> // присваивание

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

{

if (Size != mt.Size)

{

delete[]pVector;

pVector = new TVector<ValType>[Size];

Size = mt.Size;

}

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

{

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

}

return \*this;

}

***Описание алгоритма:***

Для начала производим проверку на неравенство размеров матриц, если они неравны , то удаляем исходную матрицу, заново выделяем память под новую матрицу и размер новой матрицы устанавливаем равным размеру присваиваемой матрицы, если же матрицы сразу были равны по размеру, то идём дальше без удалений и выделений памяти. И наконец, по циклу, поэлементно присваиваем матрицу , которую хотели присвоить , исходной.

* **Структура программного комплекса:**

**utmatrix.h** – файл(hader) реализации класса TMatrix и TVector

**sample\_matrix.cpp** - заголовочный файл, реализующий интерфейс пользователя

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

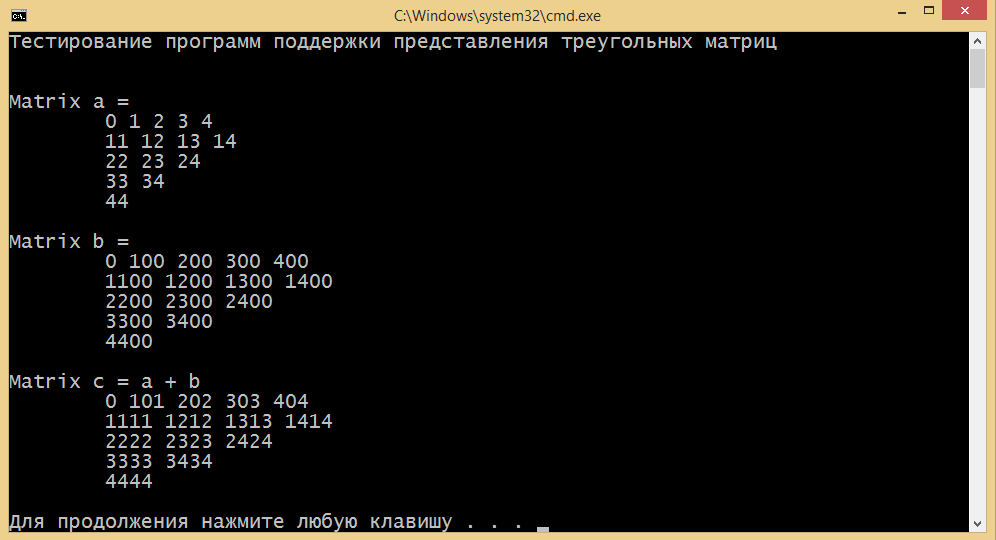
**test\_tmatrix.cpp** - тесты для класса TMatrix

## Заключение.

В результате выполнения данной лабораторной работы были реализованы:

* Класс векторов – TVector.
* Различные скалярные операции.
* Класс матриц – TMatrix.
* Различные векторные операции над матрицами.

Разработан алгоритм и составлен программный комплекс для решения соответствующей задачи.



## Список используемой литературы.

* Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования», ННГУ, 2002. – 100 c.
* <https://ru.wikipedia.org/wiki/Матрица_(математика)>.
* Беклемишева Л.А., Петрович А.Ю., Чубаров И.А. Сборник задач по аналитической геометрии и линейной алгебре: Учеб. Пособие / Под ред. Д.В. Беклемишева. – 2-е изд., перераб. – М.:ФИЗМАТЛИТ, 2004. – 496 с. – ISBN 5-9221-0010-6.

## Приложение.

#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 "can\_create\_vector\_with\_positive\_length";

}

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

{

throw "can\_create\_vector\_with\_positive\_length";

}

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

{

throw "can\_create\_vector\_with\_positive\_length";

}

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

{

if (Size != v.Size)

{

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 (Size != v.Size)

{

delete[]pVector;

pVector = new ValType[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 res(Size, StartIndex);

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

{

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

}

return res;

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

template <class ValType> // вычесть скаляр

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

{

TVector res(Size, StartIndex);

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

{

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

}

return res;

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

template <class ValType> // умножить на скаляр

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

{

TVector res(Size, StartIndex);

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

{

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

}

return res;

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

template <class ValType> // сложение

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

{

TVector res(Size, StartIndex);

if (Size != v.Size)

{

throw "cant\_add\_vectors\_with\_not\_equal\_size";

}

for (int i = 0; i < 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)

{

TVector res(Size, StartIndex);

if (Size != v.Size)

{

throw "can\_subtract\_vectors\_with\_equal\_size";

}

for (int i = 0; i < 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)

{

ValType res=0;

if (Size != v.Size)

{

throw "can\_multiply\_vectors\_with\_equal\_size";

}

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++)

{

cout << "\t";

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 "out\_of\_range";

}

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

{

pVector[i] = TVector<ValType>(s - i, i);

}

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

template <class ValType> // конструктор копирования

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

{

Size = mt.Size;

pVector = new TVector<ValType>[Size];

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

{

pVector[i] = TVector<ValType>(mt.pVector[i]);

}

}

template <class ValType> // конструктор преобразования типа

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

{

Size = mt;

pVector = new TVector<ValType>[Size];

}

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 (Size != mt.Size)

{

delete[]pVector;

pVector = new TVector<ValType>[Size];

Size = 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 res(Size);

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

{

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

}

return res;

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

template <class ValType> // вычитание

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

{

TMatrix res(Size);

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

{

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

}

return res;

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

// TVector О3 Л2 П4 С6

// TMatrix О2 Л2 П3 С3

#endif

#include <iostream>

#include "utmatrix.h"

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

void main()

{

int i, j;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование программ поддержки представления треугольных матриц"<< endl;

cout << endl;

cout << endl;

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

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

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

{

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

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

}

c = a + b;

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

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

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

}

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