Федеральное государственное автономное образовательное учреждение высшего образования **"Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ).**

**Институт информационных технологий, математики и механики**

**ОТЧЕТ**

По лабораторной работе

На тему:

«Библиотека для работы с векторами и матрицами»

**Выполнила:**

Студентка группы 3821Б1ПМ3

Киселева К.В.

**Проверил:**

Заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[Введение. 3](#_Toc104251086)

[1. Постановка задачи. 4](#_Toc104251087)

[2. Руководство пользователя. 5](#_Toc104251088)

[2.1. Библиотека «Vector.h» 5](#_Toc104251089)

[2.2. Библиотека «Matrix.h» 6](#_Toc104251090)

[3. Руководство программиста. 8](#_Toc104251091)

[Литература. 12](#_Toc104251092)

[Приложение. 13](#_Toc104251093)

[Приложение 1. 13](#_Toc104251094)

[Приложение 2. 14](#_Toc104251095)

[Приложение 3. 19](#_Toc104251096)

# Введение.

Программирование - это интересный, полезный и увлекательный процесс, благодаря которому мы, с помощью специальных команд, заставляем компьютер, выполнять для нас различные задачи, от выполнения операций с числами и навигации, до управления самолетами, спутниками и прочей техникой.

Случайные числа являются одной из основных составляющих любого языка программирования, на них строятся различные алгоритмы.

В данной лабораторной работе для изучения методов и особенностей работы со случайными числами была поставлена задача, которую нужно было выполнить, используя язык программирования «С».

# Постановка задачи.

Написать классы для работы с векторами и матрицами, использовать шаблоны. Вектора в математическом понимании: имеется набор значений из N мерного пространства, размерность задается как параметр. Классы вектора и матрицы должны быть вынесены в статическую библиотеку.

# Руководство пользователя.

Были созданы две статические библиотеки, одна – для работы с векторами, вторая ­– для работы с матрицами.

# Библиотека «Vector.h»

В этой библиотеке представлены стандартные конструкторы, деструктор, перегруженные операторы и методы.

* **Конструкторы и деструктор.** (см. Таблица 1)

|  |  |
| --- | --- |
| TVector(); | Используя этот конструктор, получим вектор, имеющий нулевую длинну и не содержащий никаких данных. |
| TVector(const int n); | Используя этот конструктор, получим вектор длины n, каждая координата которого изначально равна 0 |
| TVector(const int n, const T\* \_data); | Используя этот конструктор, получим вектор длины n, координаты будут равны соответствующим значениям из массива, который мы передаём. |
| TVector(const int \_len, const T a); | Используя этот конструктор, получим вектор длины \_len, каждая координата которого изначально равна значению числа a. |
| TVector(const TVector<T>& a); | Используя этот конструктор, получим вектор, равный вектору a. |
| ~TVector(); | Используя деструктор, мы удалим все данные о векторе. |

Таблица 1 (Конструкторы и деструктор)

* **Операторы.**

В библиотеке перегружены операторы и мы можем: складывать, вычитать, умножать и делить векторы, умножать вектор на число, присваивать векторы, проверять их равенство. Так же перегружены операторы потокового ввода и вывода, оператор индексации.

* **Методы.** (см. Таблица 2)

|  |  |
| --- | --- |
| int GetLen() const; | Метод, позволяющий получить значение длины вектора. |
| T GetCoord(const int n) const; | Метод, позволяющий получить значение координаты вектора. |
| void SetVector(const int \_len, const T\* \_data); | Метод, позволяющий задать вектор. |
| void Resize(int NewLen); | Метод, позволяющий изменить длину вектора. |
| void BubbleSort(); | Метод, сортирующий массив координат вектора пузырьком. |
| void InsertSort(); | Метод, сортирующий массив координат вектора вставкой. |
| void QuickSort(); | Метод, сортирующий массив координат вектора быстрой сортировкой. |

Таблица 2 (Методы)

# Библиотека «Matrix.h»

В этой библиотеке так же представлены стандартные конструкторы, деструктор, перегруженные операторы и методы.

* **Конструкторы и деструктор.** (см. Таблица 3)

|  |  |
| --- | --- |
| TMatrix(); | Используя этот конструктор, получим матрицу, имеющую нулевую высоту и длину, а также не содержащую данных. |
| TMatrix(const int \_rowsCount, const int \_columnsCount, const T a); | Используя этот конструктор, получим матрицу определённой высоты и длины, каждая координата которой изначально равна числу a. |
| TMatrix(const int \_columnsCount, const TVector<T>\* \_columns); | Используя этот конструктор, получим матрицу длины n, координаты будут равны вектору, который мы передали. |
| TMatrix(const int \_columnsCount, const TVector<T>& \_column); | Используя этот конструктор, получим вектор длины \_len, каждая координата которого изначально равна значению числа a. |
| TMatrix(const TMatrix<T>& obj); | Используя этот конструктор, получим матрицу, равную матрице obj. |
| TMatrix(const TVector<T>& vect); | Используя этот конструктор, получим матрицу размера вектора vect. |
| ~TVector(); | Используя деструктор, мы удалим все данные о матрице. |

Таблица 3 (Конструкторы и деструктор)

* **Операторы.**

В библиотеке перегружены операторы и мы можем: складывать, вычитать, умножать матрицы, умножать на вектор, на число, присваивать матрицы, проверять их равенство. Так же перегружены операторы потокового ввода и вывода, оператор индексации.

* **Методы.** (см. Таблица 4)

|  |  |
| --- | --- |
| T GetElem(const int row, const int column) const; | Метод, позволяющий получить значение ячейки матрицы. |
| int GetRowsCount() const; | Метод, позволяющий получить количество строк матрицы. |
| int GetColumnsCount() const; | Метод, позволяющий получить количество столбцов матрицы. |
| TVector<T> GetVector(int column) const; | Метод, позволяющий получить значение вектора (всей строки). |
| void SetElem(const int row, const int column, T a); | Метод, позволяющий задать значение в ячейке матрицы. |
| void SetMatrix(const int \_columnsCount, const TVector<T>\* \_columns); | Метод, позволяющий задать матрицу через вектор. |
| void Transp(); | Метод, позволяющий транспонировать матрицу. |

Таблица 4(Методы)

# Руководство программиста.

* 1. ***Описание структуры программы.***

Программа состоит из двух файлов «Vector.h» и «Matrix.h», в которых написаны статические библиотеки, и файла типа «main.cpp», в котором написано само приложение.

* 1. ***Описание алгоритмов.***
     1. **Алгоритм сортировки пузырьком.**

|  |
| --- |
| T temp;  for (int i = 0; i < len; i++)  {  for (int j = len - 1; j > i; j--)  {  if (data[j - 1] > data[j])  {  temp = data[j - 1];  data[j - 1] = data[j];  data[j] = temp;  }  }  } |

* + 1. **Алгоритм сортировки вставкой.**

|  |
| --- |
| T temp;  for (int i = 1; i < len; i++)  {  for (int j = i - 1; j >= 0 && data[j] > data[j + 1]; j--)  {  temp = data[j + 1];  data[j + 1] = data[j];  data[j] = temp;  }  } |

* + 1. **Алгоритм быстрой сортировки.**

|  |
| --- |
| void Quick(T\* a, int N)  {  int i = 0, j = N - 1;  T temp, p;  p = a[N >> 1];  while (i < j)  {  while (a[i] < p) i++;  while (a[j] > p) j--;  if (i < j)  {  temp = a[i];  a[i] = a[j];  a[j] = temp;  i++;  j--;  }  }  if (j > 1) Quick(a, j);  if (N > i + 1) Quick(a + i, N - i);  }  template <class T>  void TVector<T>::QuickSort()  {  Quick(data, len);  return;  } |

1. **Эксперименты.**

Проведём несколько экспериментов с разными типами сортировок.

Для упрощения понимания результатов эксперимента используем таблицу. Введём разное количество данных и сравним время работы.

|  |  |  |
| --- | --- | --- |
| **Тип сортировки** | **Размер входных данных** | **Время работы, мс** |
| Пузырьком | 10000 |  |
| Вставками | 10000 |  |
| Быстрая | 10000 |  |
| Пузырьком | 100000 |  |
| Вставками | 100000 |  |
| Быстрая | 100000 |  |

Таблица 5 (Эксперименты)

Из Таблица 5 видно, что быстрее всего работает быстрая сортировка, на 10 000 элементах она работает меньше 1 мс.

**Заключение.**

В ходе лабораторной работы на языке программирования «С++» были написаны классы для работы с векторами и матрицами, использовать шаблоны. Вектора в математическом понимании: имеется набор значений из N мерного пространства, размерность задается как параметр. Классы вектора и матрицы были вынесены в статическую библиотеку.

Литература.  
1. Т.А. Павловская Учебник по программированию на языках высокого  
уровня(С/С++) – Режим доступа: http://cph.phys.spbu.ru/documents/First/books/7.pdf  
2. Бьерн Страуструп. Язык программирования С++ - Режим доступа:  
<http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>

# Приложение.

## Приложение 1.

|  |
| --- |
| #include "Vector.h"  #include "Matrix.h"  #include <iostream>  using namespace std;  int main()  {  TVector<int> a(2, 3);  int mas[2] = { 1, -1 };  TVector<int> b(2, mas);  TVector<int> c = a + b;  TMatrix<int> Q(2, 2, 1);  TMatrix<int> W(2, 2, 1);  TMatrix<int> C = W \* a;    c.QuickSort();  cout << mas[1] << endl;  cout << c << endl;  cout << C << endl;  cout << Q \* C << endl;  } |

## Приложение 2.

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  template <class T>  class TVector  {  protected:  int len;  T\* data;  public:  TVector();  TVector(const int n);  TVector(const int n, const T\* \_data);  TVector(const int \_len, const T a);  TVector(const TVector<T>& a);  ~TVector();  TVector<T>& operator=(const TVector<T>& a);  TVector<T> operator+(const TVector<T>& a);  TVector<T> operator-(const TVector<T>& a);  TVector<T> operator\*(const T a);  T operator\*(const TVector<T>& a);  TVector<T> operator/(const TVector<T>& a);  bool operator==(const TVector<T>& a);  T& operator[](const int n);  int GetLen() const;  T GetCoord(const int n) const;  void SetVector(const int \_len, const T\* \_data);  void Resize(int NewLen);  void BubbleSort();  void InsertSort();  void QuickSort();  friend ostream& operator<<(ostream& t, const TVector<T>& a) {  if (a.GetLen() == 0)  {  t << 0 << endl;  return t;  }  for (int i = 0; i < a.GetLen() - 1; i++)  {  t << a.GetCoord(i) << ", ";  }  t << a.GetCoord(a.GetLen() - 1) << endl;  return t;  }  friend istream& operator>>(istream& t, TVector<T>& v)  {  if (v.GetLen() == 0)  throw "Error: vector == 0";  for (int i = 0; i < v.GetLen(); i++)  {  cout << "Enter vector " << i << " coordinate" << endl;  t >> v[i];  }  return t;  }  };  template <class T>  TVector<T>::TVector()  {  len = 0;  data = 0;  }  template <class T>  TVector<T>::~TVector()  {  if (data != 0)  {  delete[] data;  data = 0;  }  }  template<class T>  TVector<T>::TVector(const int n)  {  if (n >= 0)  {  len = n;  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = 0;  }  else  throw "Error, n < 0";  }  template <class T>  TVector<T>::TVector(const int \_len, const T\* \_data)  {  if ((\_len == 0) || (\_data == 0))  {  TVector();  return;  }  len = \_len;  data = new T[len];  for (int i = 0; i < len; i++)  {  data[i] = \_data[i];  }  }  template<class T>  TVector<T>::TVector(const int n, const T a)  {  if (n == 0)  {  TVector();  return;  }  if (n > 0)  {  len = n;  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = a;  }  else  throw "Error, n < 0";  }  template <class T>  TVector<T>::TVector(const TVector<T>& a)  {  if (a.len == 0)  {  TVector();  return;  }  len = a.len;  data = new T[len];  for (int i = 0; i < len; i++)  {  data[i] = a.GetCoord(i);  }  }  template<class T>  TVector<T>& TVector<T>::operator=(const TVector<T>& a)  {  len = a.len;  if (a.len == 0)  throw "Error: vector == 0";  if (data != 0)  {  delete[] data;  data = 0;  }  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = a.data[i];  return \*this;  }  template<class T>  inline TVector<T> TVector<T>::operator+ (const TVector<T>& a)  {  if (len != a.GetLen())  throw "Error, len rasnie";  TVector<T> res(len);  for (int i = 0; i < len; i++)  res[i] = this->data[i] + a.data[i];  return res;  }  template <class T>  TVector<T> TVector<T>::operator-(const TVector<T>& a)  {  return ((\*this) + a \* (-1));  }  template <class T>  TVector<T> TVector<T>::operator\*(const T a)  {  if (len == 0)  return \*this;  TVector<T> res(len);  for (int i = 0; i < len; i++)  res[i] = a \* data[i];  return res;  }  template<class T>  T TVector<T>::operator\*(const TVector<T>& a)  {  if (len != a.GetLen())  throw "Error: len rasnie";  if (len == 0)  throw "Error: len == 0";  T res = 0;  for (int i = 0; i < len; i++)  res += (\*this)[i] \* a.GetCoord(i);  return res;  }  template <class T>  TVector<T> TVector<T>::operator/(const TVector<T>& a)  {  if (len != a.GetLen())  throw "Error: len rasnie";  if (len == 0)  throw "Error: len == 0";  TVector<T> res(len);  for (int i = 0; i < len; i++)  {  if (a.GetCoord != 0)  res[i] = data[i] / a.GetCoord(i);  else res[i] = 0;  }  return res;  }  template<class T>  bool TVector<T>::operator==(const TVector<T>& a)  {  if (len != a.len)  return false;  for (int i = 0; i <= a.len; i++)  {  if (data[i] != a.data[i])  return false;  }  return true;  }  template <class T>  T& TVector<T>::operator[](const int n)  {  if ((n < 0) || (n >= len))  {  throw "Error: out of tuple range";  }  return data[n];  }  template <class T>  int TVector<T>::GetLen() const  {  return len;  }  template <class T>  T TVector<T>::GetCoord(const int n) const  {  if (n < 0 || n >= len)  throw "Error: out of tuple range";  return data[n];  }  template <class T>  void TVector<T>::SetVector(const int \_len, const T\* \_data)  {  if (\_len == 0)  {  if (data != 0)  {  delete[] data;  data = 0;  }  return;  }  len = \_len;  if (data != 0)  {  delete[] data;  data = 0;  }  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = \_data[i];  }  template<class T>  void TVector<T>::Resize(int NewLen)  {  if (NewLen < 0)  throw "Eror len < 0";  else  {  T\* t = new T[NewLen];  for (int i = 0; i < min(NewLen, len); i++)  t[i] = data[i];  delete[]data;  len = NewLen;  data = t;  }  }  template <class T>  void TVector<T>::BubbleSort()  {  T temp;  for (int i = 0; i < len; i++)  {  for (int j = len - 1; j > i; j--)  {  if (data[j - 1] > data[j])  {  temp = data[j - 1];  data[j - 1] = data[j];  data[j] = temp;  }  }  }  }  template <class T>  void TVector<T>::InsertSort()  {  T temp;  for (int i = 1; i < len; i++)  {  for (int j = i - 1; j >= 0 && data[j] > data[j + 1]; j--)  {  temp = data[j + 1];  data[j + 1] = data[j];  data[j] = temp;  }  }  }  template <class T>  void Quick(T\* a, int N)  {  int i = 0, j = N - 1;  T temp, p;  p = a[N >> 1];  while (i < j)  {  while (a[i] < p) i++;  while (a[j] > p) j--;  if (i < j)  {  temp = a[i];  a[i] = a[j];  a[j] = temp;  i++;  j--;  }  }  if (j > 1) Quick(a, j);  if (N > i + 1) Quick(a + i, N - i);  }  template <class T>  void TVector<T>::QuickSort()  {  Quick(data, len);  return;  } |

## Приложение 3.

|  |
| --- |
| #pragma once  #include "Vector.h"  template <class T>  class TMatrix : public TVector<T>  {  protected:  int rowsCount;  int columnsCount;  TVector<T>\* columns;  public:  TMatrix();  TMatrix(const int \_rowsCount, const int \_columnsCount, const T a);  TMatrix(const int \_columnsCount, const TVector<T>\* \_columns);  TMatrix(const int \_columnsCount, const TVector<T>& \_column);  TMatrix(const TMatrix<T>& obj);  TMatrix(const TVector<T>& vect);  ~TMatrix();  bool operator==(const TMatrix<T>& m);  TMatrix<T>& operator=(const TMatrix<T>& m);  TVector<T>& operator[](const int indx);  TMatrix<T> operator+(const TMatrix<T>& m);  TMatrix<T> operator-(const TMatrix<T>& m);  TMatrix<T> operator\*(const TMatrix<T>& m);  TMatrix<T> operator\*(const T a);  TVector<T> operator\*(const TVector<T>& v);  T GetElem(const int row, const int column) const;  int GetRowsCount() const;  int GetColumnsCount() const;  TVector<T> GetVector(int column) const;  void SetElem(const int row, const int column, T a);  void SetMatrix(const int \_columnsCount, const TVector<T>\* \_columns);  void Transp();  friend ostream& operator<<(ostream& t, const TMatrix<T>& m)  {  if (m.GetColumnsCount() == 0)  throw "Error: cannot print 0-matrix";  for (int i = 0; i < m.GetRowsCount(); i++)  {  for (int j = 0; j < m.GetColumnsCount(); j++)  t << m.GetElem(i, j) << ' ';  t << '\n';  }  return t;  }  friend istream& operator>>(istream& t, const TMatrix<T>& m)  {  if (m.GetColumnsCount() == 0)  throw "Error: cannot write in 0-matrix";  for (int i = 0; i < m.GetRowsCount(); i++)  {  for (int j = 0; j < m.GetColumnsCount(); j++)  {  cout << "Enter (" << j << ", " << i << ") element:" << endl;  t >> m[j][i];  }  }  return t;  }  };  template <class T>  TMatrix<T>::TMatrix()  {  rowsCount = 0;  columnsCount = 0;  columns = 0;  }  template <class T>  TMatrix<T>::~TMatrix()  {  if (columns != 0)  {  delete[] columns;  columns = 0;  }  }  template <class T>  TMatrix<T>::TMatrix(const int \_rowsCount, const int \_columnsCount, const T a)  {  if (\_rowsCount == 0 || \_columnsCount == 0)  {  TMatrix();  return;  }  rowsCount = \_rowsCount;  columnsCount = \_columnsCount;  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = TVector<T>(rowsCount, a);  }  template <class T>  TMatrix<T>::TMatrix(const int \_columnsCount, const TVector<T>\* \_columns)  {  if (\_columnsCount == 0 || \_columns == 0)  {  TMatrix();  return;  }  columnsCount = \_columnsCount;  rowsCount = \_columns[0].GetLen();  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = \_columns[i];  }  template <class T>  TMatrix<T>::TMatrix(const int \_columnsCount, const TVector<T>& \_column)  {  if (\_columnsCount == 0 || \_column.GetLen() == 0)  {  TMatrix();  return;  }  columnsCount = \_columnsCount;  rowsCount = \_column.GetLen();  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = \_column;  }  template <class T>  TMatrix<T>::TMatrix(const TMatrix<T>& obj)  {  if (obj.columnsCount == 0)  {  TMatrix();  return;  }  columnsCount = obj.columnsCount;  rowsCount = obj.rowsCount;  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = obj.columns[i];  }  template <class T>  TMatrix<T>::TMatrix(const TVector<T>& vect)  {  if (vect.GetLen() == 0)  {  TMatrix();  return;  }  columnsCount = 1;  rowsCount = vect.GetLen();  columns = new TVector<T>[1];  columns[0] = vect;  }  template <class T>  bool TMatrix<T>::operator==(const TMatrix<T>& m)  {  if (columnsCount == m.GetColumnsCount() && rowsCount == m.GetRowsCount())  {  for (int i = 0; i < columnsCount; i++)  {  if (columns[i] != m.GetVector(i))  return false;  }  return true;  }  return false;  }  template <class T>  TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& m)  {  if (m.GetColumnsCount == 0 || m.GetRowsCount() == 0)  throw "Error: matrix=0-matrix";  columnsCount = m.GetColumnsCount();  rowsCount = m.GetRowsCount();  if (columns != 0)  {  delete[] columns;  columns = 0;  }  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = m.GetVector(i);  return \*this;  }  template <class T>  TVector<T>& TMatrix<T>::operator[](const int indx)  {  if (indx < 0 || indx >= columnsCount)  throw "Error: run out of matrix range";  return columns[indx];  }  template <class T>  TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& m)  {  if (m.GetColumnsCount() != columnsCount || m.GetRowsCount() != rowsCount)  throw "Error: different matrix size in +";  TMatrix<T> temp(columnsCount, rowsCount, 0);  for (int i = 0; i < columnsCount; i++)  temp[i] = this->GetVector(i) + m.GetVector(i);  return temp;  }  template <class T>  TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& m)  {  if (m.GetColumnsCount() != columnsCount || m.GetRowsCount() != rowsCount)  throw "Error: different matrix size in +";  TMatrix<T> temp(columnsCount, rowsCount, 0);  for (int i = 0; i < columnsCount; i++)  temp[i] = this->GetVector(i) - m.GetVector();  return temp;  }  template <class T>  TMatrix<T> TMatrix<T>::operator\*(const T a)  {  TMatrix<T> temp = this;  for (int i = 0; i < columnsCount; i++)  temp[i] = temp[i] \* a;  return temp;  }  template <class T>  TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& m)  {  if (columnsCount != m.GetRowsCount())  throw "Error: cannot multiply matrix (invalid size)";  TMatrix<T> temp(m.GetColumnsCount(), rowsCount, 0);  for (int k = 0; k < rowsCount; k++)  {  for (int j = 0; j < m.GetColumnsCount(); j++)  {  for (int i = 0; i < columnsCount; i++)  temp[j][k] += this->GetElem(k, i) \* m.GetElem(i, j);  }  }  return temp;  }  template <class T>  TVector<T> TMatrix<T>::operator\*(const TVector<T>& v)  {  if (v.GetLen() != columnsCount)  throw "Error: cannot muiltiply matrix\*vector (invalid size)";  TVector<T> temp(rowsCount);  for (int i = 0; i < rowsCount; i++)  {  for (int j = 0; j < v.GetLen(); j++)  temp[i] += this->GetElem(i, j) \* v.GetCoord(j);  }  return temp;  }  template <class T>  int TMatrix<T>::GetRowsCount() const  {  return rowsCount;  }  template <class T>  int TMatrix<T>::GetColumnsCount() const  {  return columnsCount;  }  template <class T>  T TMatrix<T>::GetElem(const int row, const int column) const  {  if (row < 0 || row >= rowsCount || column < 0 || column >= columnsCount)  throw "Error: run out of matrix range";  return columns[column][row];  }  template <class T>  TVector<T> TMatrix<T>::GetVector(int column) const  {  if (column < 0 || column >= columnsCount)  throw "Error: run out of matrix range";  return columns[column];  }  template <class T>  void TMatrix<T>::SetElem(const int row, const int column, T a)  {  if (row < 0 || row >= rowsCount || column < 0 || column >= columnsCount)  throw "Error: out of matrix range";  columns[column][row] = a;  return;  }  template <class T>  void TMatrix<T>::SetMatrix(const int \_columnsCount, const TVector<T>\* \_columns)  {  if (\_columnsCount == 0 || \_columns == 0)  {  columnsCount = 0;  rowsCount = 0;  if (columns != 0)  {  delete[] columns;  columns = 0;  }  }  columnsCount = \_columnsCount;  rowsCount = \_columns[0].GetLen();  if (columns != 0)  {  delete[] columns;  columns = 0;  }  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = \_columns[i];  }  template <class T>  void TMatrix<T>::Transp()  {  if (columns == 0)  return;  int tm;  tm = columnsCount;  columnsCount = rowsCount;  rowsCount = tm;  TVector<T>\* temp = new TVector<T>[columnsCount];  for (int i = 0; i < rowsCount; i++)  {  for (int j = 0; j < columnsCount; j++)  temp[i][j] = columns[j][i];  }  delete[] columns;  columns = new TVector<T>[columnsCount];  for (int i = 0; i < columnsCount; i++)  columns[i] = temp[i];  return;  } |