Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ)  
Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Векторы. Матрицы. Создание статической библиотеки.

Выполнил:

студент института ИТММ

гр. 3821Б1ПМ3

Афанасьев К.О

Проверил:

заведующий лабораторией

суперкомпьютерных технологий и

высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2022 г.

Содержание

[1. Введение 3](#_Toc86228936)

[2. Постановка задачи 4](#_Toc86228937)

[3. Руководство пользователя 5](#_Toc86228938)

[4. Руководство программиста 6](#_Toc86228939)

[1) Описание структуры программы 6](#_Toc86228940)

[2) Описание структур данных 7](#_Toc86228941)

[3) Описание алгоритмов 8](#_Toc86228942)

[5. Эксперименты 10](#_Toc86228943)

[6. Заключение 1](#_Toc86228944)1

[7. Литература](#_Toc86228945) 12

[8. Приложение 1](#_Toc86228946) 13

# Введение

Электронно-вычислительные машины (ЭВМ) проникли во многие сферы человеческой деятельности. Использование ЭВМ позволяет переложить обработку информации на автоматические устройства, способные достаточно долго работать без участия человека и со скоростью, в несколько миллионов раз превышающей скорость обработки информации человеком.

Язык программирования является [формальным языком](https://www.hmong.press/wiki/Formal_language), содержащим [набор инструкций](https://www.hmong.press/wiki/Instruction_set_architecture), которые производят различные виды [продукции](https://www.hmong.press/wiki/Input/output). Языки программирования используются в [компьютерном программировании](https://www.hmong.press/wiki/Computer_programming) для реализации [алгоритмов](https://www.hmong.press/wiki/Algorithm).

Большинство языков программирования состоят из [инструкций](https://www.hmong.press/wiki/Machine_instruction) для [компьютеров](https://www.hmong.press/wiki/Computer). Существуют программируемые машины, которые используют набор [конкретных инструкций](https://www.hmong.press/wiki/Domain-specific_language), а не [общие языки программирования](https://www.hmong.press/wiki/General-purpose_language).

В ходе данной лабораторной работы мы научимся работать с векторами и матрицами. А также выносить это всё в статическую библиотеку.

# Постановка задачи

Написать классы для работы с векторами и матрицами, использовать при этом шаблоны. Сделать матрицу наследником вектора. Классы вектора и матрицы вынести в статическую библиотеку. Продемонстрировать всю работу на примере.

# Руководство пользователя

1. Необходимо ввести данные, которые от тебя требует программа. Сначала длину вектора, а затем координаты вектора А и вектора В

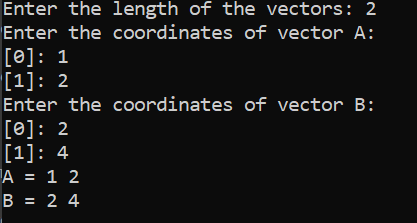


Рисунок 1. Ввод требуемых данных

1. Дальше пользователю нужно будет нажать Enter и компьютер сделает за него всё сам. Сначала выведет векторы, а после применит математические операторы

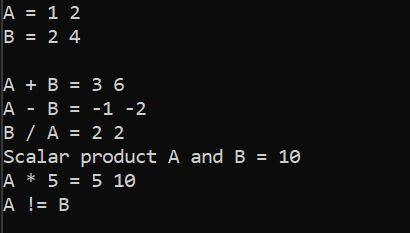


Рисунок 2. Вывод результата

# Руководство программиста

## Описание структуры программы

1.Вектор.

1.1 Cоздание шаблонного класса TVector

|  |
| --- |
| template<class T>  class TVector  {  protected:  T\* data;  int len;  public:  TVector();  TVector(int n, T v);  TVector(int n);  TVector(const TVector<T>& p);  ~TVector();  int GetLen() const;  T& operator[](int i);  TVector<T>& operator = (const TVector<T>& p);  TVector<T> operator + (const TVector<T>& p);  TVector<T> operator - (const TVector<T>& p);  TVector<T> operator / (const TVector<T>& p);  T operator\*(const TVector<T>& v);  TVector<T> operator \* (const T n);  bool operator == (const TVector<T>& p);  friend ostream& operator<<(ostream& ostr, TVector<T>& p)  {  for (int i = 0; i < p.GetLen(); i++)  ostr << " " << p[i];  if (int j = 0 <= p.GetLen()) {  cout << "\n";  j++;  }  else  {  cout << "\t ";  ++j;  }  return ostr;  }  friend istream& operator>>(istream& t, TVector<T>& v)  {  if (v.GetLen() == 0)  throw "ОШИБКА";  for (int i = 0; i < v.GetLen(); i++) {  cout << "[" << i << "]: ";  t >> v[i];  }  return t;  }  }; |

1.2 Написание конструкторов и деструкторов

|  |
| --- |
| template<class T>  inline TVector<T>::TVector()  {  len = 0;  data = nullptr;  }  template<class T>  inline TVector<T>::TVector(int n, T v)  {  if (n > 0)  {  data = new T[n];  len = n;  for (int i = 0; i < n; i++)  data[i] = v;  }  else  {  cout << "ОШИБКА" << endl;  }  }  template<class T>  inline TVector<T>::TVector(int n)  {  data = new T[n];  len = n;  for (int i = 0; i < n; i++)  data[i] = 0;  }  template<class T>  inline TVector<T>::TVector(const TVector<T>& p)  {  data = new T[p.len];  len = p.len;  for (int i = 0; i < p.len; i++)  data[i] = p.data[i];  }  template<class T>  inline TVector<T>::~TVector()  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  } |

1.3 Написание операторов +, -, \*, /, =, ==, [], потоковый ввод и вывод

|  |
| --- |
| template<class T>  inline T& TVector<T>::operator[](int i)  {  if (i >= 0 && i < len)  {  return data[i];  }  else  {  cout << "ОШИБКА";  }  }  template<class T>  inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  data = new T[p.len];  len = p.len;  for (int i = 0; i < p.len; i++)  data[i] = p.data[i];  return \*this;  }  template<class T>  inline TVector<T> TVector<T>::operator+(const TVector<T>& p)  {  if (this->len != p.len) {  cout << "ОШИБКА" << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) + p.data[q];  }  return res;  }  }  template<class T>  inline TVector<T> TVector<T>::operator-(const TVector<T>& p)  {  if (this->len != p.len) {  cout << "sizes don't match" << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) - p.data[q];  }  return res;  }  }  template<class T>  inline TVector<T> TVector<T>::operator/(const TVector<T>& p)  {  if (this->len != p.len) {  cout << "sizes don't match" << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) / p.data[q];  }  return res;  }  }  template<class T>  inline T TVector<T>::operator\*(const TVector<T>& v)  {  if (len != v.GetLen()) {  cout << "different sizes";  }  T result = 0;  for (int i = 0; i < len; i++) {  result += (this->data[i]) \* v.data[i];  }  return result;  }  template<class T>  inline TVector<T> TVector<T>::operator\*(const T n)  {  TVector<T> res(len);  for (int i = 0; i < len; i++)  res[i] = data[i] \* n;  return res;  }  template<class T>  inline bool TVector<T>::operator==(const TVector<T>& p)  {  if (len != p.len)  return false;  for (int i = 0; i < len; i++)  if (data[i] != p.data[i])  return false;  return true;  } |

Написание функции для получения длины вектора

|  |
| --- |
| template<class T>  inline int TVector<T>::GetLen() const  {  return len;  } |

1. Матрица. Наследник вектора

2.1 Создание класса.

|  |
| --- |
| template<class T>  class TMatrix : public TVector<TVector<T>>  {  protected:  int len1;  public:  TMatrix();  TMatrix(int n1);  TMatrix(int n1, int n2);  TMatrix(int n1, int n2, T v);  TMatrix(const TMatrix<T>& m);  ~TMatrix();  int GetLen1() const;  bool operator == (const TMatrix<T>& m);  TMatrix<T> operator \* (const TMatrix<T>& p);  }; |

2.2 Конструкторы

|  |
| --- |
| template<class T>  inline TMatrix<T>::TMatrix() : TVector<TVector<T>>::TVector()  {  this->len = 0;  this->len1 = 0;  this->data = 0;  }  template<class T>  inline TMatrix<T>::TMatrix(int n1) : TVector<TVector<T>>::TVector(n1)  {  len1 = 1;  for (int i = 0; i < n1; i++)  {  this->data[i] = 1;  for (int j = 0; j < 1; j++) {  data[i][j] = 0;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(int n1, int n2) : TVector<TVector<T>>::TVector(n1)  {  len1 = n2;  for (int i = 0; i < n1; i++)  {  this->data[i] = n2;  for (int j = 0; j < n2; j++) {  data[i][j] = 0;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(int n1, int n2, T v) : TVector<TVector<T>>::TVector(n1)  {  len1 = n2;  for (int i = 0; i < n1; i++)  {  this->data[i] = n2;  for (int j = 0; j < n2; j++) {  data[i][j] = v;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(const TMatrix<T>& m)  {  len1 = m.len1;  this->len = m.len;  this->data = new TVector<T>[len1];  for (int i = 0; i < len1; i++)  this->data[i] = TVector<T>(this->len);  this->len = m.len;  for (int i = 0; i < m.len1; i++)  for (int j = 0; j < m.len; j++)  this->data[i][j] = m.data[i][j];  }  template<class T>  inline TMatrix<T>::~TMatrix()  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  } |

2.3 Операторы +, -, \*, /, =, +=, -=, \*=, ==.

|  |
| --- |
| #include "Vector.h"  template<class T>  inline TMatrix <T> TMatrix <T>::operator+(const TMatrix<T>& p)  {  if (wid == p.wid && this->len == p.len)  {  TMatrix<T> res = TMatrix<T>(this->len, wid);  for (int i = 0; i < wid; i++) for (int j = 0; j < this->len; j++) res.data[i] = this->data[i] + p.data[i];  return res;  }  else throw "ОШИБКА";  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& p)  {  if (wid == p.wid && this->len == p.len)  {  TMatrix <T> res = TMatrix <T>(this->len, wid);  for (int i = 0; i < wid; i++) for (int j = 0; j < this->len; j++) res.data[i] = this->data[i] - p.data[i];  return res;  }  else throw " ОШИБКА ";  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)  {  TMatrix<T> res(p.len, len1, 0);  for (int i = 0; i < len1; i++)  {  for (int j = 0; j < p.len; j++)  {  res[i][j] = 0;  for (int l = 0; l < this->len; l++)  res[i][j] += this->data[i][l] \* p.data[l][j];  }  }  return res;  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator=(const TVector<TVector<T>& p)  {  if (this != p)  {  delete[] this->data;  wid = p.data[0].len;  this->len = p.len;  this->data = new TVector<T>[wid];  for (int i = 0; i < wid; i++) this->data[i] = TVector<T>(p.data[i]);  return \*this;  }  else throw " ОШИБКА ";  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator+=(const TMatrix<T>& p)  {  if (wid == p.wid && this->len == p.len)  {  wid = p.wid;  this->len = p.len;  for (int i = 0; i < wid; i++) this->data[i] += p.data[i];  return \*this;  }  else throw " ОШИБКА ";  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator-=(const TMatrix<T>& p)  {  if (wid == p.wid && this->len == p.len)  {  wid = p.wid;  this->len = p.len;  for (int i = 0; i < wid; i++) this->data[i] -= p.data[i];  return \*this;  }  else throw " ОШИБКА ";  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator\*=(const TMatrix<T>& p)  {  if (p.wid > 0 && p.len > 0 && p.wid == this->len)  {  TVector<T>\* \_data = new TVector<T>[p.wid](this->len);  for (int i = 0; i < wid; i++) {  for (int j = 0; j < this->len; j++) {  for (int n = 0; n < p.wid; n++) {  \_data[i][j] += this->data[i][n] \* p.data[n][j];  }  }  }  delete[] this->data;  this->data = new TVector<T>[p.wid](this->len);  for (int i = 0; i < p.wid; i++) { this->data[i] = TVector<T>(\_data[i]); }  wid = p.wid;  return \*this;  }  else throw " ОШИБКА ";  }  template<class T>  inline bool TMatrix<T>::operator==(const TMatrix<T>& m)  {  if (this->len != m.len || len1 != m.len1) return false;  for (int i = 0; i < len1; i++)  {  for (int j = 0; j < len; j++)  if (data[i][j] != m.data[i][j]) return false;  }  return true;  } |

## Описание структур данных

В программе пишется класс TMatrix для дальнейшей работы

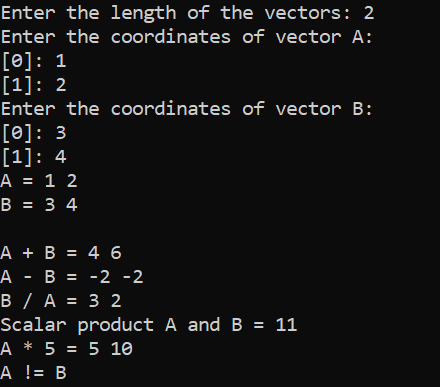
|  |
| --- |
| class TMatrix : public TVector<TVector<T>>  {  protected:  int len1;  public:  TMatrix();  TMatrix(int n1);  TMatrix(int n1, int n2);  TMatrix(int n1, int n2, T v);  TMatrix(const TMatrix<T>& m);  int GetLen1() const;  TMatrix<T>& operator=(const TMatrix<T>& p);  TMatrix<T>& operator=(const TVector<TVector<T>>& p);  TMatrix<T> operator+(const TMatrix<T>& p);  TMatrix<T> operator-(const TMatrix<T>& p);  TMatrix<T>& operator+=(const TMatrix<T>& p);  TMatrix<T>& operator-=(const TMatrix<T>& p);  TMatrix<T>& operator\*=(const TMatrix<T>& p);  bool operator == (const TMatrix<T>& m);  TMatrix<T> operator\*(const TMatrix<T>& p); |

В программе пишется класс TVector для дальнейшей работы

|  |
| --- |
| class TVector  {  protected:  T\* data;  int len;  public:  TVector();  TVector(int n = 0);  TVector(const TVector& p);  ~TVector();  int GetLen() const;  T& operator[](int i);  TVector& operator= (const TVector& p);  TVector operator+ (const TVector& p);  TVector operator- (const TVector& p);  TVector operator/ (const TVector& p);  T operator\*(const TVector& v);  TVector operator \* (const T n);  bool operator == (const TVector& p);  friend istream& operator>>(istream& in, TVector& v)  {  for (int i = 0; i < v.len; i++)  {  in >> v.data[i];  }  return in;  }  friend ostream& operator<<(ostream& out, const TVector& p)  {  for (int i = 0; i < p.len; i++)  out << p.data[i] << ' ';  return out;  }  }; |

# Эксперименты

Проведем эксперимент. Введём длину вектора, а затем координаты. Программа нам считает сумму, разность, частное и скалярное произведение.



# Заключение

В ходе лабораторной работы была написаны классы вектора и матрицы, которые впоследствии были вынесены в статическую библиотеку. В этих классах реализованы основные математические операторы. Также продемонстрирована их работа на примере.

Я научился выносить классы в статическую библиотеку, а также стал лучше писать классы и их функции на языке программирования С++. Стал больше понимать в написании конструкторов и деструкторов.

# Литература

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ — 2-е. — М.: Вильямс, 2005. — 1296 с.
2. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.
3. Язык С в ХХI веке. Бен Клеменс
4. Изучаем программирование на C. Дэвид Гриффитс, Дон Гриффитс
5. C. Полное руководство. Герберт Шилдт Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.
6. Бен Клеменс "Язык С в XXI веке
7. Дэвид Гриффитс, Дон Гриффитс "Изучаем программирование на C"

# Приложение 1

1.Vector.h

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  template<class T>  class TVector  {  protected:  T\* data;  int len;  public:  TVector();  TVector(int n, T v);  TVector(int n);  TVector(const TVector<T>& p);  ~TVector();  int GetLen() const;  T& operator[](int i);  TVector<T>& operator = (const TVector<T>& p);  TVector<T> operator + (const TVector<T>& p);  TVector<T> operator - (const TVector<T>& p);  TVector<T> operator / (const TVector<T>& p);  T operator\*(const TVector<T>& v);  TVector<T> operator \* (const T n);  bool operator == (const TVector<T>& p);  friend ostream& operator<<(ostream& ostr, TVector<T>& p)  {  for (int i = 0; i < p.GetLen(); i++)  ostr << " " << p[i];  if (int j = 0 <= p.GetLen()) {  cout << "\n";  j++;  }  else  {  cout << "\t ";  ++j;  }  return ostr;  }  friend istream& operator>>(istream& t, TVector<T>& v)  {  if (v.GetLen() == 0)  throw "ОШИБКА";  for (int i = 0; i < v.GetLen(); i++) {  cout << "[" << i << "]: ";  t >> v[i];  }  return t;  }  };  template<class T>  inline TVector<T>::TVector()  {  len = 0;  data = nullptr;  }  template<class T>  inline TVector<T>::TVector(int n, T v)  {  if (n > 0)  {  data = new T[n];  len = n;  for (int i = 0; i < n; i++)  data[i] = v;  }  else  {  cout << "ОШИБКА" << endl;  }  }  template<class T>  inline TVector<T>::TVector(int n)  {  data = new T[n];  len = n;  for (int i = 0; i < n; i++)  data[i] = 0;  if (n > 0)  {  data = new T[n];  len = n;  for (int i = 0; i < n; i++)  data[i] = 0;  }  else  {  cout << "length <= 0" << endl;  } }  template<class T>  inline TVector<T>::TVector(const TVector<T>& p)  {  data = new T[p.len];  len = p.len;  for (int i = 0; i < p.len; i++)  data[i] = p.data[i];  }  template<class T>  inline TVector<T>::~TVector()  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  }  template<class T>  inline int TVector<T>::GetLen() const  {  return len;  }  template<class T>  inline T& TVector<T>::operator[](int i)  {  if (i >= 0 && i < len)  {  return data[i];  }  else  {  cout << "ОШИБКА";  }  }  template<class T>  inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  data = new T[p.len];  len = p.len;  for (int i = 0; i < p.len; i++)  data[i] = p.data[i];  return \*this;  }  template<class T>  inline TVector<T> TVector<T>::operator+(const TVector<T>& p)  {  if (this->len != p.len) {  cout << "ОШИБКА" << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) + p.data[q];  }  return res;  }  }  template<class T>  inline TVector<T> TVector<T>::operator-(const TVector<T>& p)  {  if (this->len != p.len) {  cout << " ОШИБКА " << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) - p.data[q];  }  return res;  }  }  template<class T>  inline TVector<T> TVector<T>::operator/(const TVector<T>& p)  {  if (this->len != p.len) {  cout << "sizes don't match" << endl;  }  else {  TVector<T> res(\*this);  for (int q = 0; q < len; q++)  {  res.data[q] = (this->data[q]) / p.data[q];  }  return res;  }  }  template<class T>  inline T TVector<T>::operator\*(const TVector<T>& v)  {  if (len != v.GetLen()) {  cout << " ОШИБКА ";  }  T result = 0;  for (int i = 0; i < len; i++) {  result += (this->data[i]) \* v.data[i];  }  return result;  }  template<class T>  inline TVector<T> TVector<T>::operator\*(const T n)  {  TVector<T> res(len);  for (int i = 0; i < len; i++)  res[i] = data[i] \* n;  return res;  }  template<class T>  inline bool TVector<T>::operator==(const TVector<T>& p)  {  if (len != p.len)  return false;  for (int i = 0; i < len; i++)  if (data[i] != p.data[i])  return false;  return true;  } |

2. Matrix.h

|  |
| --- |
| #pragma once  #include "Vector.h"  template<class T>  class TMatrix : public TVector<TVector<T>>  {  protected:  int len1;  public:  TMatrix();  TMatrix(int n1);  TMatrix(int n1, int n2);  TMatrix(int n1, int n2, T v);  TMatrix(const TMatrix<T>& m);  ~TMatrix();  int GetLen1() const;  bool operator == (const TMatrix<T>& m);  TMatrix<T> operator \* (const TMatrix<T>& p);  TMatrix<T> operator \* (const TVector<T>& v);  };  template<class T>  inline TMatrix<T>::TMatrix() : TVector<TVector<T>>::TVector()  {  this->len = 0;  this->len1 = 0;  this->data = 0;  }  template<class T>  inline TMatrix<T>::TMatrix(int n1) : TVector<TVector<T>>::TVector(n1)  {  len1 = 1;  for (int i = 0; i < n1; i++)  {  this->data[i] = 1;  for (int j = 0; j < 1; j++) {  data[i][j] = 0;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(int n1, int n2) : TVector<TVector<T>>::TVector(n1)  {  len1 = n2;  for (int i = 0; i < n1; i++)  {  this->data[i] = n2;  for (int j = 0; j < n2; j++) {  data[i][j] = 0;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(int n1, int n2, T v) : TVector<TVector<T>>::TVector(n1)  {  len1 = n2;  for (int i = 0; i < n1; i++)  {  this->data[i] = n2;  for (int j = 0; j < n2; j++) {  data[i][j] = v;  }  }  }  template<class T>  inline TMatrix<T>::TMatrix(const TMatrix<T>& m)  {  len1 = m.len1;  this->len = m.len;  this->data = new TVector<T>[len1];  for (int i = 0; i < len1; i++)  this->data[i] = TVector<T>(this->len);  this->len = m.len;  for (int i = 0; i < m.len1; i++)  for (int j = 0; j < m.len; j++)  this->data[i][j] = m.data[i][j];  }  template<class T>  inline TMatrix<T>::~TMatrix()  {  if (data != nullptr)  {  delete[] data;  data = nullptr;  }  }  template<class T>  inline int TMatrix<T>::GetLen1() const  {  return len1;  }  template<class T>  inline bool TMatrix<T>::operator==(const TMatrix<T>& m)  {  if (this->len != m.len || len1 != m.len1)  return false;  for (int i = 0; i < len1; i++)  {  for (int j = 0; j < len; j++)  if (data[i][j] != m.data[i][j])  return false;  }  return true;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)  {  TMatrix<T> res(p.len, len1, 0);  for (int i = 0; i < len1; i++)  {  for (int j = 0; j < p.len; j++)  {  res[i][j] = 0;  for (int l = 0; l < this->len; l++)  res[i][j] += this->data[i][l] \* p.data[l][j];  }  }  return res;  }  template<class T>  TMatrix<T> TMatrix<T>::operator\*(const TVector<T>& v)  {  TVector<T> temp(v);  if ((this->len) != temp.GetLen()) throw "ОШИБКА";  TMatrix<T> res(1, len1, 0);  for (int i = 0; i < len1; i++)  {  for (int j = 0; j < 1; j++)  {  res[i][j] = 0;  for (int x = 0; x < this->len; x++)  res[i][j] += this->data[i][x] \* temp[x];  }  }  return res;  } |