Федеральное государственное автономное образовательное учреждение высшего образования

**«Национальный исследовательский Нижегородский государственный университет**

**им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

Направление подготовки: Прикладная математика и информатика

Отчёт по лабораторной работе

**Векторы**

**Выполнил:** студент группы 3821Б1ПМ3

Благодеров В.А.

**Проверил:** заведующий лабораторией

суперкомпьютерных технологий и

высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2021 г.

Содержание

Введение 3

1. Постановка задачи 4

2. Руководство пользователя 5

3. Руководство программиста 6

a. Описание структуры программы 6

b. Описание структуры данных 7

c. Описание алгоритмов 7

4. Эксперименты 11

Заключение 14

Литература 15

Приложения 16

Приложение 1 16

Приложение 2 16

Приложение 3 20

Приложение 4 25

Введение

Программирование – процесс создания компьютерных программ, написание инструкций на конкретном языке программирования. Программы способствуют улучшению и облегчения человеческого быта, промышленной деятельности, а также сферы услуг. Для упрощения программирования пользователь использует массивы. Массивы широко используются при решении задач из линейной алгебры, математического моделирования, в машинном обучении и т. д. Решение этих задач, в общем виде, основывается на математических операциях с использованием матриц и векторов, которые позволяют компактно записывать в виде простой формулы очень сложные преобразования. Специальные типы данных matrix и vector позволяют писать код, приближённый к математической записи, и избавляют от необходимости создавать вложенные циклы. Программисту уже не нужно помнить о правильной индексации массивов, которые участвуют в вычислении.

1. Постановка задачи

Написать классы для работы с векторами и матрицами, использовать шаблоны. Векторы в математическом понимании: имеется набор значений из N мерного пространства, размерность задаётся как параметр. Матрица должна быть наследником вектора. Классы векторы и матрицы должны быть вынесены в статическую библиотеку. Продемонстрировать их работу на примере (написать в main пример).

Должны быть:

Конструкторы (по умолчанию, инициализатор, копирования), деструктор, доступ к защищенным полям;

Перегруженные операции: +, -, \*, /, =, ==, [], потоковый ввод и вывод;

Перегруженные операции: +, -, \*, / должны быть реализованы для векторов (вектор +-\*/вектор), матриц (матрица +-\*матрица), матрично-векторные (матрица\*вектор);

1. Руководство пользователя

Классы в C++ - это абстракция описывающая методы, свойства, ещё не существующих объектов. Объекты – конкретное представление абстракции, имеющее свои свойства и методы. Чтобы воспользоваться программой необходимо создать объекты из класса «TVector» или «TMatrix». Для создания объекта применяются конструкторы. По сути конструкторы представляют специальные методы, которые называются так же как и класс, и которые вызываются при создании нового объекта и выполняют инициализацию объекта.

|  |
| --- |
| TVector(); // Конструктор по умолчанию для вектора  TVector(int n, T c); // Конструктор для вектора длины n  TMatrix(); // Конструктор по умолчанию для матрицы  TMatrix(int n); // Конструктор для квадратной матрицы n\*n  TMatrix(int l, int w); // Конструктор для прямоугольной матрицы w\*l |

Фрагмент кода 1. Конструкторы

|  |
| --- |
| TVector<int> A(5, 4); // Создали вектор длины 5, заполненный цифрами 4  TMatrix<int> m1(2, 4); // Создали матрицу длины 2 и ширины 4 |

Фрагмент кода 2. Пример создания объектов

После создания объекта можно воспользоваться функционалом класса. Для обращения к функциональности класса – полям, методам применяется точечная нотация точки – после объекта класса ставится точка, а затем элемент класса: объект.поле\_класса или объект.метод\_класса (параметры\_метода). Кроме этого C++ позволяет организовать перегрузку операций. Механизм перегрузки операций позволяет обеспечить более традиционную и удобную запись действий над объектами.

|  |
| --- |
| TVector<int> A(5, 4); // Создали вектор A  TVector<int> C(5, 4); // Создали вектор C  TVector<int> D(5, 2); // Создали вектор D  C = A + D; // В вектор C записали сумму векторов  std::cout << C << std::endl; // Вывели вектор C на экран |

Фрагмент кода 3. Пример пользования программой

1. Руководство программиста
   1. Описание структуры программы

Программа состоит из двух проектов vector и vectorLib (Библиотека). Проект vector использует главную функцию main(). Проект vectorLib содержит описание двух классов «TVector» и «TMatrix». Функции класса «TVector»:

|  |
| --- |
| TVector(); // Конструктор по умолчанию  ~TVector(); // Деструктор  TVector(int n, T c); // Конструктор инициализатор  TVector(const TVector<T>& p); // Конструктор копирования  TVector<T> operator + (const TVector<T>& p); // Перегруженный оператор +  TVector<T> operator - (const TVector<T>& p); // Перегруженный оператор -  TVector<T> operator \* (const TVector<T>& p); // Перегруженный оператор \*  TVector<T> operator / (const TVector<T>& p); // Перегруженный оператор /  TVector<T>& operator = (const TVector<T>& p); // Перегруженный оператор =  bool operator != (const TVector<T>& p); // Перегруженный оператор !=  T& operator[](int i); // Перегруженный оператор []  int GetLen(); // Возвращает длину  void Resize(int len\_); // Изменяет размер  int Count(T c); // Поиск количества вхождений символа  int\* AllCount(T c); // Все вхождения заданного символа |

Фрагмент кода 4. Методы класса «TVector»

Функции класса «TMatrix»:

|  |
| --- |
| TMatrix(); // Конструктор по умолчанию  ~TMatrix(); // Деструктор  TMatrix(int n); // Конструктор инициализатор  TMatrix(int l, int w); // Конструктор инициализатор  TMatrix(const TMatrix<T>& p); // Конструктор копирования  TMatrix(const TVector<TVector<T>>& p); // Конструктор копирования  TMatrix<T> operator + (const TMatrix<T>& p); // Перегруженный оператор +  TMatrix<T> operator - (const TMatrix<T>& p); // Перегруженный оператор -  TMatrix<T> operator \* (const TMatrix<T>& p); // Перегруженный оператор \*  TMatrix<T> operator \* (const TVector<T>& p); // Перегруженный оператор \*  TMatrix<T>& operator = (const TMatrix<T>& p); // Перегруженный оператор =  TMatrix<T>& operator = (const TVector<TVector<T>>& p); // Перегруженный оператор =  bool operator == (const TMatrix<T>& p); // Перегруженный оператор ==  void Resize(int n, int w); // Изменяет размер  TVector<T>& operator[](int i); // Перегруженный оператор +  int GetWidth(); // Возвращает ширину  int CountM(T c); // Поиск количества вхождений символа  int\*\* AllCountM(T c); // Все вхождения заданного символа |

Фрагмент кода 5. Методы класса «TMatrix»

Алгоритм работы программы: создаём объект, производим операции и выводим результат.

* 1. Описание структуры данных

Классы «TVector» и «TMatrix». В классах используются шаблоны.

* 1. Описание алгоритмов

Программа реализует различные операции над векторами и матрицами. Рассмотрим некоторые из них.

1. Сложение векторов (operator +). Суть метода. Функция принимает вектор, далее выполняется тело функции. Возвращается новый вектор (Смотри блок-схему 1).

TVector<T> res(\*this);

((len == p.len) && (p.data != 0) && (data != 0))

i=0, len

res.data[i] = res.data[i] + p.data[i];

res

Да

Нет

Блок-схема 1. «Сложение векторов»

1. Присвоение векторов (operator =). Суть метода. Функция принимает вектор, далее выполняется тело функции. Возвращается ссылка на вектор, а не сам вектор (Смотри блок-схему 2).

(this == &p)

(p.data == 0)

delete[] data;

data = 0;

len = 0;

(data != 0)

len = p.len;

data = new T[len];

i=0, len

data[i] = p.data[i];

\*this

\*this

\*this

Да

Нет

Да

Нет

Да

Блок-схема 2. «Присвоение векторов»

1. Умножение матриц (operator \*). Суть метода. Функция принимает матрицу, далее выполняется тело функции. Возвращается матрица (Смотри блок-схему 3).

(this->len == p.width)

TMatrix<T> res(\*this);

i=0, width

res[i][j] = 0;

res[i][j] += this->data[i][k] \* p.data[k][j];

res

j=0, p.len

k=0, p.len

Да

Блок-схема 3. «Умножение матриц»

1. Равенство матриц (operator ==). Суть метода. Принимает матрицу, далее выполняется тело функции. Возвращается значение 0 или 1, где 0 – ложь, а 1 – истина (Смотри блок-схему 4).

Блок-схема 4. «Оператор == для матриц»

((this->len != p.len) || (width != p.width))

i=0, width

(this->data[i][j] != p.data[i][j])

false

false

true

j=0, len

Да

Нет

Да

Нет

1. Эксперименты

Проведём несколько экспериментов и проверим работоспособность библиотеки. Создадим пару объектов и выполним над нами операции.

|  |
| --- |
| int d = 0, g = 0;  int\* q = 0;  int\*\* h = 0;  TVector<int> A(5, 4);  TVector<int> B(5, 5);  TVector<int> C(6, 3);  TVector<int> D(5, 2);  TVector<TVector<int>> j(6, B);  TMatrix<int> m1(2, 4);  TMatrix<int> m2(2);  TMatrix<int> m3(4);  TMatrix<int> m4(5); |

Фрагмент кода 6. Создание

1. Сложим вектора A и D (Рис. 1).

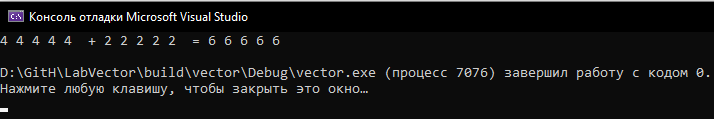


Рис. 1. Сложение векторов

1. Проверим равенство векторов A и C (Рис. 2).

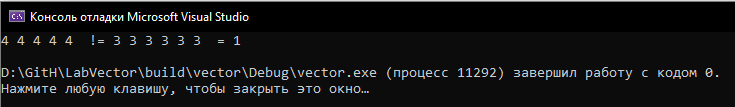


Рис. 2. Равенство векторов

1. Перемножим матрицу m4 и вектор B (Рис. 3).

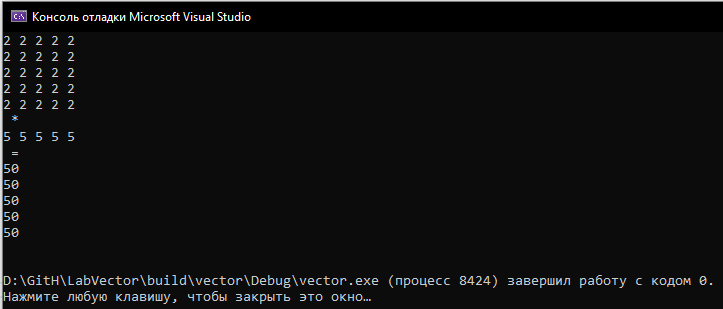


Рис. 3. Умножение матрицы на вектор

1. Перемножим матрицы m3 и m1 (Рис. 4).

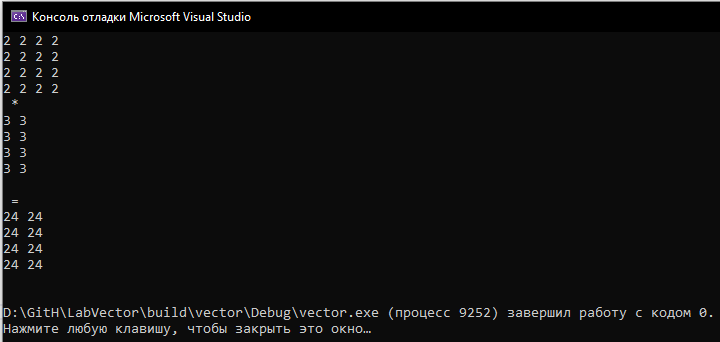


Рис. 4. Умножение матриц

1. Выведем все вхождения «3» в вектор B (Рис. 5).

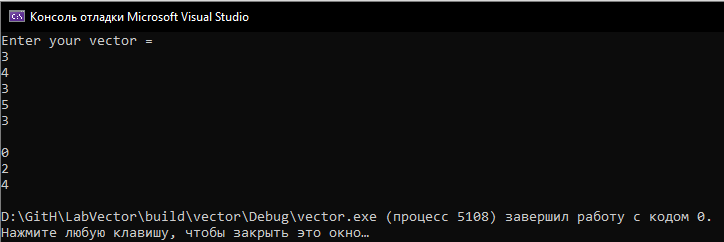


Рис. 5. Все вхождения в вектор

1. Выведем все вхождения «2» в матрицу m2 (Рис. 6).

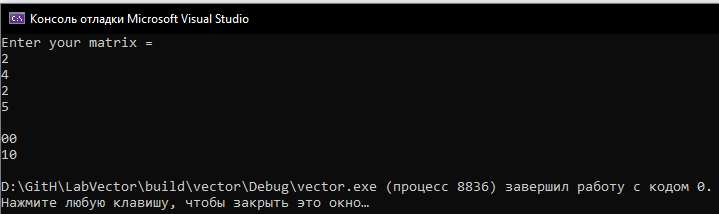


Рис. 6. Все вхождения в матрицу

Заключение

В ходе лабораторной работы мною была выполнена поставленная задача. Мне удалось написать классы для работы с векторами и матрицами, используя шаблоны. Классы векторов и матриц внесены в статическую библиотеку. Вектор является более безопасным типом данных, чем массивы. Работа с векторами во многом похожа на работу с массивами. Нумерация элементов начинается с нуля, а при выходе за границу возникает ошибка. Вектора можно создавать из элементов любого типа данных. Вектор является частным случаем матрицы. Можно сказать, что матрица – это массив векторов одинакового размера.

Литература

1. Bjarne Stroustrup: The C++ Programming Language. Addison-Wesley. 1986.

Приложения

Приложение 1

|  |
| --- |
| #include <iostream>  #include "Vector.h"  #include "Matrix.h"  int main()  {  int d = 0, g = 0;  int\* q = 0;  int\*\* h = 0;  TVector<int> A(5, 4);  TVector<int> B(5, 5);  TVector<int> C(6, 3);  TVector<int> D(5, 2);  TVector<TVector<int>> j(6, B);  TMatrix<int> m1(2, 4);  TMatrix<int> m2(2);  TMatrix<int> m3(4);  TMatrix<int> m4(5);  std::cout << A << " + " << D << " = " << A + D << std::endl;  std::cout << A << " != " << C << " = " << A.operator!=(C) << std::endl;  std::cout << m4 << " \* \n" << B << "\n = \n" << m4 \* B << std::endl;  std::cout << m3 << " \* \n" << m1 << "\n = \n" << m3 \* m1 << std::endl;  std::cin >> B;  std::cout << "\n";  d = B.Count(3);  q = B.AllCount(3);  for (int k = 0; k < d; k++)  std::cout << q[k] << std::endl;  std::cin >> m2;  std::cout << "\n";  g = m2.CountM(2);  h = m2.AllCountM(2);  for (int w = 0; w < g; w++)  std::cout << h[w][0] << h[w][1] << std::endl;  return 0;  } |

Приложение 2

|  |
| --- |
| #pragma once  #include <iostream>  template <class T>  class TVector  {  public:  TVector();  ~TVector();  TVector(int n, T c);  TVector(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);  TVector<T> operator / (const TVector<T>& p);  TVector<T>& operator = (const TVector<T>& p);  bool operator != (const TVector<T>& p);  T& operator[](int i);  int GetLen();  void Resize(int len\_);  int Count(T c);  int\* AllCount(T c);  protected:  T\* data;  int len;  };  template<class T>  inline TVector<T>::TVector()  {  data = 0;  len = 0;  }  template<class T>  inline TVector<T>::~TVector()  {  if (data != 0)  {  delete[] data;  data = 0;  }  len = 0;  }  template<class T>  inline TVector<T>::TVector(int n, T c)  {  if (n < 0) throw "n<0";  data = new T[n];  len = n;  for (int i = 0; i < len; i++)  data[i] = c;  }  template<class T>  inline TVector<T>::TVector(const TVector<T>& p)  {  len = p.len;  if (p.len == 0)  data = 0;  else  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = p.data[i];  }  template<class T>  inline TVector<T> TVector<T>::operator+(const TVector<T>& p)  {  TVector<T> res(\*this);  if ((len == p.len) && (p.data != 0) && (data != 0))  {  for (int i = 0; i < len; i++)  res.data[i] = res.data[i] + p.data[i];  }  return res;  }  template<class T>  inline TVector<T> TVector<T>::operator-(const TVector<T>& p)  {  TVector<T> res(\*this);  if ((len == p.len) && (p.data != 0) && (data != 0))  {  for (int i = 0; i < len; i++)  res.data[i] = res.data[i] - p.data[i];  }  return res;  }  template<class T>  inline TVector<T> TVector<T>::operator\*(const TVector<T>& p)  {  TVector<T> res(\*this);  if ((len == p.len) && (p.data != 0) && (data != 0))  {  for (int i = 0; i < len; i++)  res.data[i] = res.data[i] \* p.data[i];  }  return res;  }  template<class T>  inline TVector<T> TVector<T>::operator/(const TVector<T>& p)  {  TVector<T> res(\*this);  if ((len == p.len) && (p.data != 0) && (data != 0))  {  for (int i = 0; i < len; i++)  {  if (p.data[i] == 0) throw "delenie na 0";  res.data[i] = res.data[i] / p.data[i];  }  }  return res;  }  template<class T>  inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)  {  if (this == &p) return \*this;  if (p.data == 0)  {  delete[] data;  data = 0;  len = 0;  return \*this;  }  if (data != 0) delete[] data;  len = p.len;  data = new T[len];  for (int i = 0; i < len; i++)  data[i] = p.data[i];  return \*this;  }  template<class T>  inline bool TVector<T>::operator!=(const TVector<T>& p)  {  if (len != p.len) return true;  else  {  for (int i = 0; i < len; i++)  if (data[i] != p.data[i])  return true;  }  return false;  }  template<class T>  inline T& TVector<T>::operator[](int i)  {  if (len == 0) throw "not found";  if ((i < 0) || (i > len)) throw "not found";  return data[i];  }  template<class T>  inline int TVector<T>::GetLen()  {  return len;  }  template<class T>  inline void TVector<T>::Resize(int len\_)  {  T\* res = 0;  if (len\_ <= 0) throw "no";  else  {  T\* res = new T[len\_];  for (int i = 0; i < len\_; i++)  res[i] = data[i];  if (data != 0)  delete[] data;  len = len\_;  data = res;  }  }  template<class T>  inline int TVector<T>::Count(T c)  {  int k = 0;  for (int i = 0; i < len; i++)  if (data[i] == c) k++;  return k;  }  template<class T>  inline int\* TVector<T>::AllCount(T c)  {  int count = 0;  int k = 0;  for (int i = 0; i < len; i++)  if (data[i] == c) k++;  int\* q = new int[k];  for (int j = 0; j < len; j++)  if (data[j] == c)  {  q[count] = j;  count++;  }  return q;  }  template <class T>  std::ostream& operator<<(std::ostream& ostr, TVector<T>& p)  {  for (int i = 0; i < p.GetLen(); i++)  ostr << p[i] << " ";  return ostr;  }  template <class T>  std::istream& operator>>(std::istream& istr, TVector<T>& p)  {  T a = 0;  std::cout << "Enter your vector = " << std::endl;  for (int i = 0; i < p.GetLen(); i++)  {  istr >> a;  p[i] = a;  }  return istr;  } |

Приложение 3

|  |
| --- |
| #pragma once  #include "Vector.h"  template <class T>  class TMatrix : public TVector<TVector<T>>  {  protected:  int width;  public:  TMatrix();  ~TMatrix();  TMatrix(int n);  TMatrix(int l, int w);  TMatrix(const TMatrix<T>& p);  TMatrix(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 TVector<T>& p);  TMatrix<T>& operator = (const TMatrix<T>& p);  TMatrix<T>& operator = (const TVector<TVector<T>>& p);  bool operator == (const TMatrix<T>& p);  void Resize(int n, int w);  TVector<T>& operator[](int i);  int GetWidth();  int CountM(T c);  int\*\* AllCountM(T c);  };  template<class T>  inline TMatrix<T>::TMatrix()  {  this->data = 0;  width = 0;  this->len = 0;  }  template<class T>  inline TMatrix<T>::~TMatrix()  {  if (this->data != 0)  delete[] data;  data = 0;  this->len = 0;  width = 0;  }  template<class T>  inline TMatrix<T>::TMatrix(int n)  {  if (n < 0) throw "n<0";  this->len = n;  width = n;  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = TVector<T>(len, 2);  }  template<class T>  inline TMatrix<T>::TMatrix(int l, int w)  {  width = w;  this->len = l;  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = TVector<T>(l, 3);  }  template<class T>  inline TMatrix<T>::TMatrix(const TMatrix<T>& p)  {  if (p.data == 0)  {  TMatrix();  return;  }  this->len = p.len;  this->width = p.width;  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = p.data[i];  }  template<class T>  inline TMatrix<T>::TMatrix(const TVector<TVector<T>>& p)  {  TVector<TVector<T>> a = p;  width = a.GetLen();  this->len = a[0].GetLen();  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = a[i];  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& p)  {  TMatrix<T> res(\*this);  if ((this->len == p.len) && (width == p.width))  {  for (int i = 0; i < width; i++)  res[i] = this->data[i] + p.data[i];  }  return res;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& p)  {  TMatrix<T> res(\*this);  if ((this->len == p.len) && (width == p.width))  {  for (int i = 0; i < width; i++)  res[i] = this->data[i] - p.data[i];  }  return res;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)  {  if (this->len == p.width)  {  TMatrix<T> res(p.len, width);  for (int i = 0; i < width; i++)  for (int j = 0; j < p.len; j++)  {  res[i][j] = 0;  for (int k = 0; k < width; k++)  res[i][j] += this->data[i][k] \* p.data[k][j];  }  return res;  }  else throw "umnojenie ne robit";  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator\*(const TVector<T>& p)  {  TVector<T> tmp = p;  int len\_ = tmp.GetLen();  if (this->len == len\_)  {  TMatrix<T> res(\*this);  res.Resize(1, len\_);  for (int i = 0; i < width; i++)  for (int j = 0; j < 1; j++)  {  res[i][j] = 0;  for (int k = 0; k < len\_; k++)  res[i][j] += this->data[i][k] \* tmp[k];  }  return res;  }  else throw "umnojenie ne robit";  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& p)  {  if (this->data != 0)  delete[] data;  this->len = p.len;  width = p.width;  if (p.data == 0)  this->data = 0;  else  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = p.data[i];  return \*this;  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator=(const TVector<TVector<T>>& p)  {  if (this->data != 0)  delete[] data;  TVector<TVector<T>> a = p;  width = a.GetLen();  this->len = a[0].GetLen();  this->data = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->data[i] = a[i];  return \*this;  }  template<class T>  inline bool TMatrix<T>::operator==(const TMatrix<T>& p)  {  if ((this->len != p.len) || (width != p.width))  return false;  else  {  for (int i = 0; i < width; i++)  for (int j = 0; j < len; j++)  {  if (this->data[i][j] != p.data[i][j])  return false;  }  return true;  }  }  template<class T>  inline void TMatrix<T>::Resize(int n, int w)  {  if ((n < 0) || (w < 0)) throw "error";  else  {  TVector<T>\* temp = new TVector<T>[w];  for (int i = 0; i < w; i++)  {  temp[i] = TVector<T>(n, w);  temp[i].Resize(n);  for (int j = 0; j < n; j++)  temp[i][j] = this->data[i][j];  }  if (this->data != 0)  delete[] data;  this->len = n;  this->width = w;  this->data = temp;  }  }  template<class T>  inline TVector<T>& TMatrix<T>::operator[](int i)  {  return this->data[i];  }  template<class T>  inline int TMatrix<T>::GetWidth()  {  return width;  }  template<class T>  inline int TMatrix<T>::CountM(T c)  {  int k = 0;  for (int i = 0; i < width; i++)  for (int j = 0; j < len; j++)  if (data[i][j] == c) k++;  return k;  }  template<class T>  inline int\*\* TMatrix<T>::AllCountM(T c)  {  int count = 0;  int k = 0;  for (int i = 0; i < width; i++)  for (int j = 0; j < len; j++)  if (data[i][j] == c) k++;  int\*\* q = new int\* [k];  for (int h = 0; h < k; h++)  q[h] = new int[2];  for (int i = 0; i < width; i++)  for (int j = 0; j < len; j++)  if (data[i][j] == c)  {  q[count][0] = i;  q[count][1] = j;  count++;  }  return q;  }  template <class T>  std::ostream& operator<<(std::ostream& ostr, TMatrix<T>& p)  {  for (int i = 0; i < p.GetWidth(); i++)  ostr << p[i] << std::endl;  return ostr;  }  template <class T>  std::istream& operator>>(std::istream& istr, TMatrix<T>& p)  {  T a = 0;  std::cout << "Enter your matrix = " << std::endl;  for (int i = 0; i < p.GetWidth(); i++)  for (int j = 0; j < p.GetLen(); j++)  {  istr >> a;  p[i][j] = a;  }  return istr;  } |

Приложение 4

|  |
| --- |
| #include "Vector.h"  #include "Matrix.h" |