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

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Векторы и матрицы

Выполнил:

студент ф-та ИТММ гр. 3821Б1ФИ3

Лупша Е.А

Проверил:

Заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[**Введение** 3](#_Toc103941085)

[**1.** **Постановка задачи** 4](#_Toc103941086)

[**2.** **Руководство пользователя** 5](#_Toc103941087)

[**3.** **Руководство программиста** 6](#_Toc103941088)

[***3.1.*** ***Описание структуры программы*** 6](#_Toc103941089)

[***3.2.*** ***Описание структур данных*** 6](#_Toc103941090)

[***3.3.*** ***Описание алгоритмов*** 10](#_Toc103941091)

[3.3.1. Алгоритм сложения матриц: 10](#_Toc103941092)

[3.3.2. Алгоритм умножения матриц: 11](#_Toc103941093)

[3.3.3. Алгоритм сортировки пузырьком: 12](#_Toc103941094)  
 3.3.4. Алгоритм быстрой сортировки: 13  
 3.3.5. Алгоритм сортировки вставками: 14

[**4.** **Эксперименты** 1](#_Toc103941095)5

[**5.** **Заключение** 1](#_Toc103941096)6

[**6.** **Литература** 1](#_Toc103941097)7

[**7.** **Приложения** 1](#_Toc103941098)8

[**7.1. Приложение 1 (main.cpp)** 1](#_Toc103941099)8

[**7.2. Приложение 2 (vector.h)** 2](#_Toc103941100)0

[**7.3. Приложение 3 (matrix.h)** 2](#_Toc103941101)5

# **Введение**

Распределение по классам, есть везде и язык программирования С++ не является исключением. Классы являются основой С++. Эта структура данных позволяет группировать данные и методы внутри одной переменной. В данной лабораторной работе мы рассмотрим классы векторов и матриц, их методы, а так же классическое применение механизма перегрузки стандартных операций. Данная программа позволит нам провести работу с объектами линейной алгебры.

1. **Постановка задачи**

Написать шаблоны классов матрицы и вектора.

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 и наоборот);
* сортировка вектора тремя способами (пузырьком, вставками,

быстрая(Хоара)).

1. **Руководство пользователя**

При запуске программы нас встречает готовый созданный вектор типа int, для которого требуется ввести 4 целых числа.

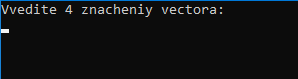


Рисунок 1. Задание значений вектора

После введённых значений, на экран выведутся некоторые операции над векторами и их результаты вычислений, а так же сортировки вектора заполненного случайными числами.

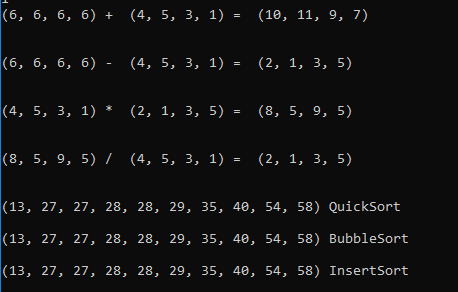


Рисунок 2. Пример операций над векторами

Далее программа предлагает ввести 6 значений для того, чтобы заполнить матрицу размера 2х3.

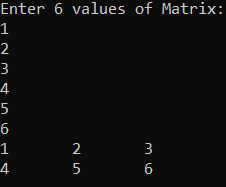


Рисунок 3. Пример заполнения матрицы

После заполнения матрицы, программа выведет на экран операции над матрицами с векторами, после чего, в случае правильного ввода чисел, программа будет завершена.

1. **Руководство программиста**
   1. ***Описание структуры программы***

Программа состоит из одного решения, в котором проект vector и vectorLib.

В решении Lab4 определено 3 модуля: main.cpp, Matrix.h, Vector.h.

* В модуле main.cpp определена стандартная функция int main(), где происходит работа с остальными модулями.
* В модуле Matrix.h определен класс TMatrix, а также объявлены все его методы и их определения.
* В модуле Vector.h определен класс TVector, а также объявлены все его методы и их определения.
  1. ***Описание структур данных***

В программе определены два следующих шаблонных класса:

Class TMatrix

Class TVector

Внутри класса TVector определены следующие поля:

* T\* vector – указатель;
* int length – количество элементов вектора;

Внутри класса TVector определен следующий набор public-методов:

1. Конструктор по умолчанию

|  |
| --- |
| TVector(); |

1. Конструктор инициализатор 1

|  |
| --- |
| TVector(int a, T \_vector); |

1. Конструктор копирования

|  |
| --- |
| TVector(const TVector<T>& p); |

1. Деструктор

|  |
| --- |
| ~TVector(); |

1. Перегрузка операции +

|  |
| --- |
| TVector<T> operator +(const TVector<T>& p); |

1. Перегрузка операции –

|  |
| --- |
| TVector<T> operator -(const TVector<T>& p); |

1. Перегрузка операции \*

|  |
| --- |
| TVector<T> operator \*(const TVector<T>& p); |

1. Перегрузка операции /

|  |
| --- |
| TVector<T> operator /(const TVector<T>& p); |

1. Перегрузка операции =

|  |
| --- |
| TVector<T>& operator =(const TVector<T>& p); |

1. Перегрузка операции ==

|  |
| --- |
| bool operator ==(const TVector<T>& p); |

1. Перегрузка операции !=

|  |
| --- |
| bool operator !=(const TVector<T>& p); |

1. Перегрузка операции индексации

|  |
| --- |
| T& operator [](int i); |

1. Метод задания длины вектора

|  |
| --- |
| void SetLen(int p); |

1. Метод получения длины вектора

|  |
| --- |
| int GetLength(); |

1. Метод, который позволяет отсортировать вектор пузырьком

|  |
| --- |
| void BubbleSort(); |

1. Метод, который позволяет отсортировать вектор вставками

|  |
| --- |
| void InsertSort(); |

1. Метод, который позволяет отсортировать вектор сортировкой Хоара

|  |
| --- |
| void QuickSort(int left, int right); |

1. Перегрузка операции потокового вывода

|  |
| --- |
| template <class T>  friend std::ostream& operator << (std::ostream& B, TVector<T>& A); |

1. Перегрузка операции потокового ввода

|  |
| --- |
| template <class T>  friend std::istream& operator >> (std::istream& B, TVector<T>& A); |

Внутри класса TMatrix определены следующие поля:

* int width – количество строк матрицы (ширина);
* int index – количество чисел, входящих в матрицу для удобного вывода (для дополнительного задания);

Внутри класса TMatrix определен следующий набор public-методов:

1. Конструктор по умолчанию

|  |
| --- |
| TMatrix(); |

1. Конструктор инициализатор 1

|  |
| --- |
| TMatrix(int l); |

1. Конструктор инициализатор 2

|  |
| --- |
| TMatrix(int w, int l); |

1. Конструктор копирования

|  |
| --- |
| TMatrix(const TMatrix<T>& p); |

1. Деструктор

|  |
| --- |
| ~TMatrix(); |

1. Перегрузка операции +

|  |
| --- |
| TMatrix<T> operator + (const TMatrix<T>& p); |

1. Перегрузка операции –

|  |
| --- |
| TMatrix<T> operator - (const TMatrix<T>& p); |

1. Перегрузка операции \* (для матриц)

|  |
| --- |
| TMatrix<T> operator \* (const TMatrix<T>& p); |

1. Перегрузка операции \* (для умножения матрицы на вектор)

|  |
| --- |
| TVector<T> operator \* (TVector<T>& p); |

1. Перегрузка операции =

|  |
| --- |
| TMatrix<T>& operator = (const TMatrix<T>& p); |

1. Перегрузка операции ==

|  |
| --- |
| bool operator == (const TMatrix<T>& p); |

1. Перегрузка операции !=

|  |
| --- |
| bool operator != (const TMatrix<T>& p); |

1. Перегрузка операции индексации

|  |
| --- |
| TVector<T>& operator [] (const int i); |

1. Метод получения количества строк матрицы

|  |
| --- |
| int GetWidth(); |

1. Метод задания количества строк матрицы

|  |
| --- |
| void SetWidth(int w); |

1. Перегрузка операции потокового вывода

|  |
| --- |
| template <class T>  friend std::ostream& operator << (std::ostream& B, TMatrix<T>& A); |

1. Перегрузка операции потокового ввода

|  |
| --- |
| template <class T>  friend std::istream& operator >> (std::istream& B, TMatrix<T>& A); |

Также в модуле Matrix.h реализована шаблонная функция умножения вектора на матрицу:

|  |
| --- |
| template <class T>  TMatrix<T> operator \*(Tvector<T>& vector, Tmatrix<T>& matrix); |

* 1. ***Описание алгоритмов***

### 3.3.1. Алгоритм сложения матриц:

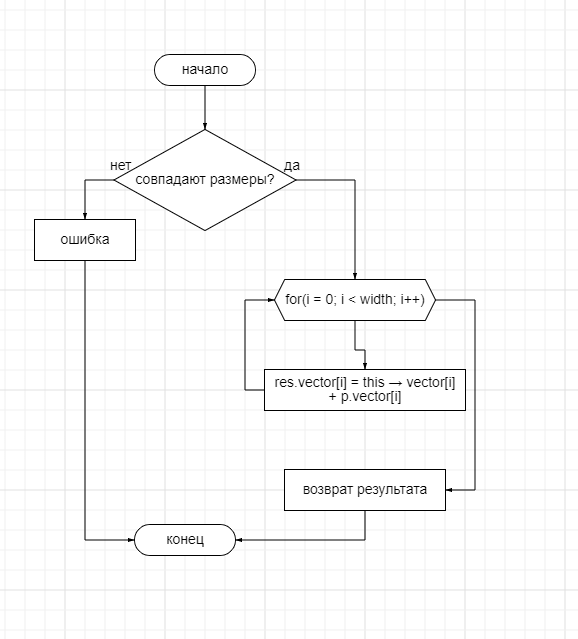


Рисунок 4. Сложение матриц блок-схема

### 3.3.2. Алгоритм умножения матриц:

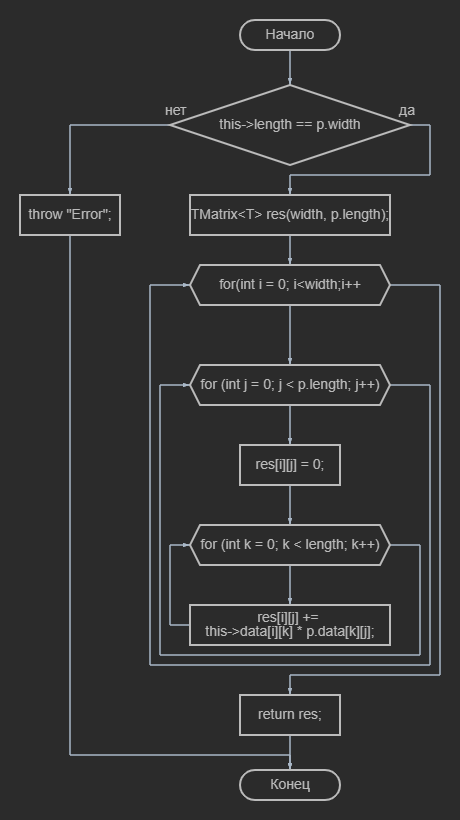


Рисунок 5. Умножение матриц блок-схема

### 3.3.3. Алгоритм сортировки пузырьком:

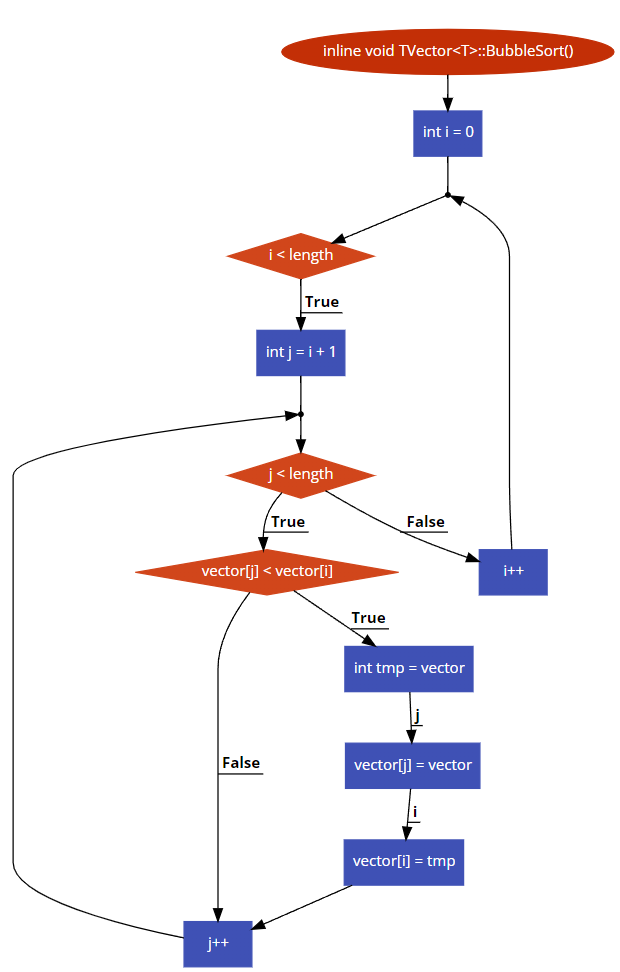


Рисунок 6. Сортировка пузырьком

### 3.3.4. Алгоритм быстрой сортировки (общий вид):

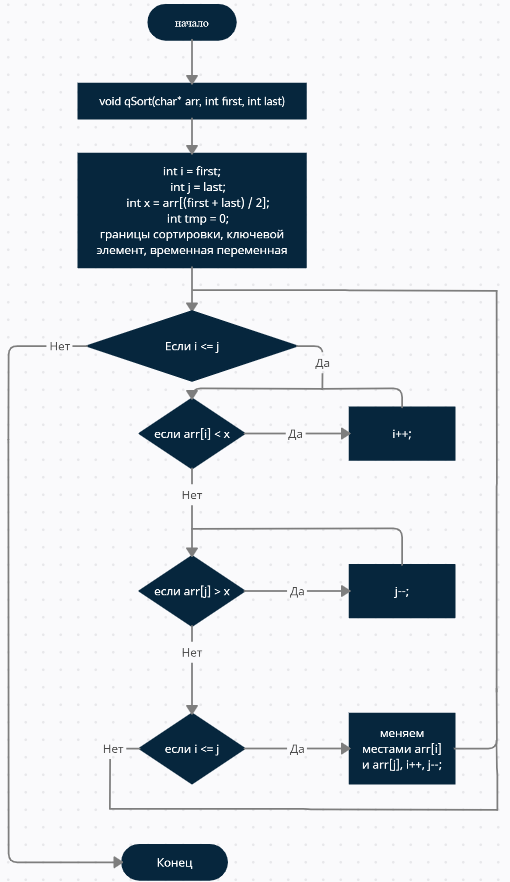


Рисунок 7. Быстрая сортировка (общий вид)

### 3.3.4. Алгоритм сортировки вставками:

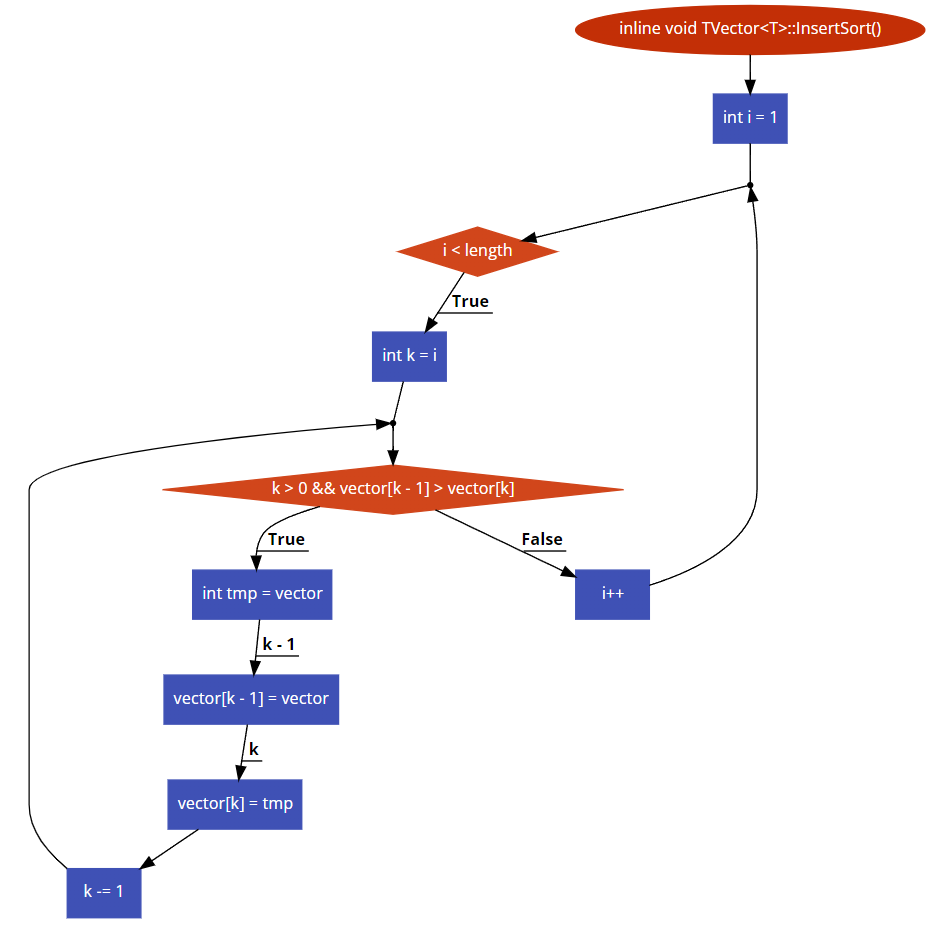


Рисунок 8. Сортировка вставками

1. **Эксперименты**
2. Оценим время, которое нужно для сложений матриц, асимптотическая сложность которого *O()*:

Таблица 1. Результаты работы сложения матриц.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек). |
| 1000 | 0.22 |
| 2000 | 0.75 |
| 4000 | 2.3 |

При увеличении количества элементов, мы видим, что время увеличивается приблизительно в 4 раза.

1. Оценим время, которое нужно для умножения матриц, асимптотическая сложность которого *O()*:

Таблица 2. Результаты работы умножения матриц.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек). |
| 1000 | 7.3 |
| 2000 | 52.1 |
| 4000 | 615.3 |

При увеличении количества элементов, мы видим, что время увеличивается приблизительно в 8 раз.

1. Сортировки векторов:

Таблица 3. Результаты работы сортировок.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | 100 000 | 250 000 | 500 000 | 1 000 000 |
| min : max | 0:255 | 0:255 | 0:255 | 0:255 |
| Сорт. Пузырьком, сек | 10.232 | 63.414 | 253.931 | N/A |
| Сорт. Вставками, сек | 8.395 | 52.66 | 219.639 | N/A |
| Быстрая сорт., сек | 0.008 | 0.021 | 0.041 | 0.084 |

1. **Заключение**

Удалось реализовать полноценную работу матриц и векторов, а также операции над ними.

Написаны шаблоны классов матрицы и вектора.

Продемонстрирована их работу на примере (в main написан пример).

В программе есть:

* конструкторы (по умолчанию, инициализатор, копирования), деструктор, доступ к защищенным полям;
* перегруженные операции: +, - , \* ,= ,== , [] потоковый ввод и вывод;
* перегруженные операции +, - ,\* реализованы для векторов (вектор +-\* вектор), матриц (матрица +-\* матрица), матрично-векторные (матрица \* вектор и наоборот);
* сортировка вектора тремя способами (пузырьком, вставками,

быстрая(Хоара)).

1. **Литература**
2. Лекции Карпенко Сергея Николаевича , 2022.
3. Практики и лабораторные Лебедева Ильи Геннадьевича, 2022.
4. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.
5. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ — 2-е. — М.: Вильямс, 2005. — 1296 с.
6. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / под ред. В. Т. Тертышного (гл. 5) и. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с.
7. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.
8. **Приложения**

**7.1. Приложение 1 (main.cpp)**

|  |
| --- |
| #include "Vector.h"  #include <clocale>  #include <malloc.h>  #include <vector>  #include <iostream>  #include <iomanip>  #include <ctime>  #include "Matrix.h"  using namespace std;  void line()  {  std::cout << "\n\n";  }  int main()  {  // vectora  TVector<int> vec1(4, 6);  TVector<int> vec2(4, 2);  TVector<int> vec3(vec2);  cin >> vec2;  vec3 = vec1 + vec2;  cout << "vec3 = vec1 + vec2 =\n" << vec1 << "+ " << vec2 << "= " << vec3 << "\n";  line();  vec3 = vec1 - vec2;  cout << "vec3 = vec1 - vec2 =\n" << vec1 << "- " << vec2 << "= " << vec3 << "\n";  line();  vec1 = vec2 \* vec3;  cout << "vec1 = vec2 \* vec3 =\n" << vec2 << "\* " << vec3 << "= " << vec1 << "\n";  line();  vec3 = vec1 / vec2;  cout << "vec3 = vec1 / vec2 =\n" << vec1 << "/ " << vec2 << "= " << vec3 << "\n";  line();  // sortirovki  TVector<int> vec11(10, 5);  srand(time(NULL));  for (int i = 0; i < vec11.GetLength(); i++)  {  vec11[i] = rand() % 50 + 10;  }  vec11.QuickSort();  std::cout << vec11 << "QuickSort";  line();  TVector<int> vec12(10, 5);  srand(time(NULL));  for (int i = 0; i < vec12.GetLength(); i++)  {  vec12[i] = rand() % 50 + 10;  }  vec12.BubbleSort();  std::cout << vec12 << "BubbleSort";  line();  TVector<int> vec13(10, 5);  srand(time(NULL));  for (int i = 0; i < vec13.GetLength(); i++)  {  vec13[i] = rand() % 50 + 10;  }  vec13.InsertSort();  std::cout << vec13 << "InsertSort";  line();  // matrix  TMatrix<int> Matrix1;  TMatrix<int> Matrix2(2, 3);  TMatrix<int> Matrix3(Matrix2);  TMatrix<int> Matrix4(4, 5);  int\*\* indexes;  std::cin >> Matrix2;  std::cout << Matrix2;  line();  for (int i = 0; i < Matrix3.GetWidth(); i++)  for (int j = 0; j < Matrix3.GetLength(); j++)  Matrix3[i][j] = rand();  cout << "Matrix3 = \n" << Matrix3;  line();  Matrix1 = Matrix2 + Matrix3;  cout << "Matrix1 = Matrix2 + Matrix3 =\n"  << Matrix2 << "\n + \n\n" << Matrix3 << "\n = \n\n" << Matrix1 << "\n";  line();  Matrix2 = Matrix3 - Matrix1;  cout << "Matrix2 = Matrix3 - Matrix1 =\n"  << Matrix3 << "\n - \n\n" << Matrix1 << "\n = \n\n" << Matrix2 << "\n";  line();  Matrix1.Resize(2, 2);  Matrix2.Resize(2, 2);  Matrix3.Resize(2, 2);  Matrix3 = Matrix2 \* Matrix1;  cout << "Matrix3 = Matrix2 \* Matrix1 =\n"  << Matrix2 << "\n \* \n\n" << Matrix1 << "\n = \n\n" << Matrix3 << "\n";  line();  for (int i = 0; i < Matrix4.GetWidth(); i++)  for (int j = 0; j < Matrix4.GetLength(); j++)  Matrix4[i][j] = rand() % 50;  vec1 = Matrix4 \* vec2;  cout << "vec1 = Matrix4 \* vec2 = \n" << Matrix4 << "\n \*\n\n" << vec2 << "\n\n =\n\n" << vec1;  line();  } |

**7.2. Приложение 2 (vector.h)**

|  |
| --- |
| #pragma once  #include <iostream>  template <class T>  class TVector  {  public:  TVector();  TVector(int a, T \_vector);  TVector(const TVector<T>& p);  ~TVector();  void SetLen(int p);  int GetLength();  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);  bool operator !=(const TVector<T>& p);  T& operator [](int i);  // sortirovki  void BubbleSort();  void InsertSort();  void QuickSort(int left = 0, int right = 0);  template <class T>  friend std::ostream& operator << (std::ostream& B, TVector<T>& A);  template <class T>  friend std::istream& operator >> (std::istream& B, TVector<T>& A);  protected:  T\* vector;  int length;  };  template<class T>  inline TVector<T>::TVector()  {  length = 0;  vector = nullptr;  }  template<class T>  inline TVector<T>::TVector(int a, T \_vector)  {  if (a > 0)  {  vector = new T[a];  length = a;  for (int i = 0; i < length; i++)  vector[i] = \_vector;  }  else throw "Error";  }  template<class T>  inline TVector<T>::TVector(const TVector<T>& p)  {  if (p.vector == nullptr)  {  vector = nullptr;  length = 0;  }  else  {  length = p.length;  vector = new T[length];  for (int i = 0; i < length; i++)  vector[i] = p.vector[i];  }  }  template<class T>  inline TVector<T>::~TVector()  {  delete[] vector;  vector = nullptr;  length = 0;  }  template<class T>  inline void TVector<T>::SetLen(int p)  {  length = p.length;  }  template<class T>  inline int TVector<T>::GetLength()  {  return length;  }  template<class T>  inline TVector<T> TVector<T>::operator+(const TVector<T>& p)  {  if (length == 0) throw "Error";  if (length != p.length) throw "Error, vectors with different dimension";  TVector<T> R(length, 0);  for (int i = 0; i < length; i++)  R[i] = vector[i] + p.vector[i];  return R;  }  template<class T>  inline TVector<T> TVector<T>::operator-(const TVector<T>& p)  {  if (length == 0) throw "Error";  if (length != p.length) throw "Error";  TVector<T> R(length, 0);  for (int i = 0; i < length; i++)  R[i] = vector[i] - p.vector[i];  return R;  }  template<class T>  inline TVector<T> TVector<T>::operator\*(const TVector<T>& p)  {  if (length == 0) throw "Error";  if (length != p.length) throw "Error";  TVector<T> R(length, 0);  for (int i = 0; i < length; i++)  R[i] = vector[i] \* p.vector[i];  return R;  }  template<class T>  inline TVector<T> TVector<T>::operator/(const TVector<T>& p)  {  if (length == 0) throw "Error";  if (length != p.length) throw "Error";  TVector<T> R(length, 0);  for (int i = 0; i < length; i++)  if (p.vector[i] == 0) R[i] = 0;  else R[i] = vector[i] / p.vector[i];  return R;  }  template<class T>  inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)  {  if (this == &p) return \*this;  if (vector != nullptr) delete[] vector;  if (p.vector == nullptr)  {  length = 0;  vector = nullptr;  }  else  {  length = p.length;  vector = new T[length];  for (int i = 0; i < length; i++)  vector[i] = p.vector[i];  }  return \*this;  }  template<class T>  inline bool TVector<T>::operator==(const TVector<T>& p)  {  if (length != p.length) return false;  for (int i = 0; i < length; i++)  if (vector[i] != p.vector[i]) return false;  return true;  }  template<class T>  inline bool TVector<T>::operator!=(const TVector<T>& p)  {  if (length != p.length) return true;  for (int i = 0; i < length; i++)  if (vector[i] != p.vector[i]) return true;  return false;  }  template<class T>  inline T& TVector<T>::operator[](int i)  {  if (vector == nullptr)  throw "Error, length <= 0";  if (i < 0 || i >= length)  throw "Error";  return vector[i];  }  template<class T>  inline void TVector<T>::BubbleSort()  {  for (int i = 0; i < length; i++)  for (int j = i + 1; j < length; j++)  if (vector[j] < vector[i])  {  int tmp = vector[j];  vector[j] = vector[i];  vector[i] = tmp;  }  }  template<class T>  inline void TVector<T>::InsertSort()  {  for (int i = 1; i < length; i++)  {  int k = i;  while (k > 0 && vector[k - 1] > vector[k])  {  int tmp = vector[k - 1];  vector[k - 1] = vector[k];  vector[k] = tmp;  k -= 1;  }  }  }  template<class T>  inline void TVector<T>::QuickSort(int left, int right)  {  int i, j;  if (right == 0)  {  left = 0;  right = length - 1;  }  i = left;  j = right;  int sr = vector[(left + right) / 2];  int tmp;  while (i <= j) {  while (vector[i] < sr) i++;  while (vector[j] > sr) j--;  if (i <= j) {  tmp = vector[i];  vector[i] = vector[j];  vector[j] = tmp;  i++;  j--;  }  }  if (left < j) QuickSort(left, j);  if (right > i) QuickSort(i, right);  }  template<class T>  inline std::ostream& operator<<(std::ostream& B, TVector<T>& A)  {  std::cout << "(";  for (int i = 0; i < A.GetLength(); i++)  {  if (i + 1 != A.GetLength())  B << A[i] << ", ";  else  {  B << A[i];  }    }  std::cout << ") ";  return B;  }  template<class T>  inline std::istream& operator>>(std::istream& B, TVector<T>& A)  {  std::cout << "Vvedite " << A.length << " znacheniy vectora: \n";  for (int i = 0; i < A.length; i++)  B >> A[i];  return B;  } |

**7.3. Приложение 3 (matrix.h)**

|  |
| --- |
| #pragma once  #include "Vector.h"  template <class T>  class TMatrix: public TVector<TVector<T>>  {  public:  TMatrix();  TMatrix(int a);  TMatrix(int w, int l);  TMatrix(const TMatrix<T>& p);  ~TMatrix();  TMatrix<T> operator + (const TMatrix<T>& p);  TMatrix<T> operator - (const TMatrix<T>& p);  TMatrix<T> operator \* (const TMatrix<T>& p);  TVector<T> operator \* (TVector<T>& p);  TMatrix<T>& operator = (const TMatrix<T>& p);  bool operator == (const TMatrix<T>& p);  bool operator != (const TMatrix<T>& p);  TVector<T>& operator [] (const int i);  int GetWidth();  void SetWidth(int w);  void Resize(int w, int l);  int GetIndex();  template <class T>  friend std::ostream& operator << (std::ostream& B, TMatrix<T>& A);  template <class T>  friend std::istream& operator >> (std::istream& B, TMatrix<T>& A);  protected:  int width;  int index;  };  template<class T>  inline TMatrix<T>::TMatrix()  {  index = 0;  width = 0;  this->vector = nullptr;  }  template<class T>  inline TMatrix<T>::TMatrix(int a) : TMatrix()  {  if (a < 0) throw "Error";  this->length = a;  width = a;  this->vector = new TVector<T>[width];  for (int i = 0; i < length; i++)  this->vector[i].Resize(length);  }  template<class T>  inline TMatrix<T>::TMatrix(int w, int l) : TMatrix()  {  if (l < 0 || w < 0) throw "Error";  this->length = l;  width = w;  this->vector = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->vector[i].Resize(length);  }  template<class T>  inline TMatrix<T>::TMatrix(const TMatrix<T>& p)  {  this->length = p.length;  width = p.width;  index = p.index;  if (p.vector == nullptr) this->vector = nullptr;  else  {  this->vector = new TVector<T>[width];  for (int i = 0; i < width; i++)  this->vector[i] = p.vector[i];  }  }  template<class T>  inline TMatrix<T>::~TMatrix()  {  if (vector != nullptr)  {  delete[] vector;  vector = nullptr;  }  width = 0;  index = 0;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& p)  {  if (this->length != p.length || width != p.width) throw "Error";  TMatrix<T> res(\*this);  for (int i = 0; i < width; i++)  res.vector[i] = this->vector[i] + p.vector[i];  return res;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& p)  {  if (this->length != p.length || width != p.width) throw "Error";  TMatrix<T> res(width, this->length);  for (int i = 0; i < width; i++)  res.vector[i] = this->vector[i] - p.vector[i];  return res;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)  {  if (this->length != p.width) throw "Error";  TMatrix<T> res(width, p.length);  for (int i = 0; i < width; i++)  {  for (int j = 0; j < p.length; j++)  {  res[i][j] = 0;  for (int k = 0; k < length; k++)  {  res[i][j] += this->vector[i][k] \* p.vector[k][j];  }  }  }  return res;  }  template<class T>  inline TVector<T> TMatrix<T>::operator\*(TVector<T>& p)  {  if (this->length != p.GetLength()) throw "Error";  TVector<T> res(width, 0);  for (int i = 0; i < width; i++)  {  for (int j = 0; j < this->length; j++)  {  res[i] += this->vector[i][j] \* p[j];  }  }  return res;  }  template<class T>  inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& p)  {  if (this == &p) return \*this;  this->length = p.length;  this->width = p.width;  if (this->vector != nullptr) delete[] vector;  if (p.vector == nullptr) vector = nullptr;  else vector = new TVector<T>[p.width];  for (int i = 0; i < p.width; i++)  this->vector[i] = p.vector[i];  return \*this;  }  template<class T>  inline bool TMatrix<T>::operator==(const TMatrix<T>& p)  {  if (this->length != p.length || width != p.width) return false;  for (int i = 0; i < width; i++)  for (int j = 0; j < this->length; j++)  if (this->vector[i][j] != this->p.vector[i][j]) return false;  return true;  }  template<class T>  inline bool TMatrix<T>::operator!=(const TMatrix<T>& p)  {  if (this->length != p.length || width != p.width) return true;  for (int i = 0; i < width; i++)  for (int j = 0; j < this->length; j++)  if (this->vector[i][j] != this->p.vector[i][j]) return true;  return false;  }  template<class T>  inline TVector<T>& TMatrix<T>::operator[](const int i)  {  if (i < 0 || i >= this->width) throw "Error";  return this->vector[i];  }  template<class T>  inline int TMatrix<T>::GetWidth()  {  return width;  }  template<class T>  inline void TMatrix<T>::SetWidth(int w)  {  this->Resize(this->length, w);  }  template<class T>  inline void TMatrix<T>::Resize(int w, int l)  {  TVector<T>\* vector1 = new TVector<T>[w];  if (this->vector != nullptr)  {  int minW, minL;  if (width > w) minW = w;  else minW = width;  if (this->length > l) minL = l;  else minL = this->length;  for (int i = 0; i < minW; i++)  {  vector1[i].Resize(l);  for (int j = 0; j < minL; j++)  vector1[i][j] = this->vector[i][j];  }  delete[] vector;  }  this->vector = vector1;  this->length = l;  width = w;  }  template<class T>  inline int TMatrix<T>::GetIndex()  {  return index;  }  template<class T>  inline std::ostream& operator<<(std::ostream& B, TMatrix<T>& A)  {  for (int i = 0; i < A.width; i++)  {  for (int j = 0; j < A.length; j++)  B << A[i][j] << "\t";  B << "\n";  }  return B;  }  template<class T>  inline std::istream& operator>>(std::istream& B, TMatrix<T>& A)  {  std::cout << "Enter " << A.width \* A.length << " values of Matrix: \n";  for (int i = 0; i < A.width; i++)  for (int j = 0; j < A.length; j++)  B >> A.vector[i][j];  return B;  }  template <class T>  TMatrix<T> operator \*(TVector<T>& vector, TMatrix<T>& matrix)  {  if (matrix.GetWidth() != 1) throw "Error";  TMatrix<T> res(vector.GetLength(), matrix.GetLength());  for (int i = 0; i < vector.GetLength(); i++)  {  for (int j = 0; j < matrix.GetLength(); j++)  {  res[i][j] = vector[i] \* matrix[0][j];  }  }  return res;  }; |