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

Выполнил:

студент ф-та ИТММ гр. 3821Б1ФИ3

Кухарева.A.P.

Проверил:

зав. лабораторией СТиВВ

Лебедев И.Г.

Нижний Новгород

2022 г

**Оглавление**

[**Введение** 3](#_30j0zll)

[**1.**](#_1fob9te) **Постановка задачи** 4

[**2.**](#_3znysh7) **Руководство пользователя** 5

[**3.**](#_2et92p0) **Руководство программиста** 6

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

[***3.2.***](#_3dy6vkm) ***Описание структур данных*** 6

[***3.3.***](#_1t3h5sf) ***Описание алгоритмов*** 10

[**3.3.1. Алгоритм суммирования матриц: 10**](#_4d34og8)

[**4.**](#_3rdcrjn) **Эксперименты** 11

[**5.**](#_26in1rg) **Заключение** 12

[**6.**](#_lnxbz9) **Литература** 13

[**7.**](#_35nkun2) **Приложения** 14

[**7.1. Приложение 1 (main.cpp)** 1](#_1ksv4uv)4

[**7.2. Приложение 2 (vector.h)** 1](#_44sinio)5

[**7.3. Приложение 3 (matrix.h)** 2](#_2jxsxqh)0

# **Введение**

Векторы и матрицы в С++ являются одномерными и двумерными массивами чисел, над которыми в данной лабораторной работе предстоит выполнить операции сложения, умножения, и другие математические действия.

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 нужно ввести 5 целых чисел.



Рисунок 1. Ввод значений вектора.

После того, как мы ввели все 5 значений вектора, на экран выводится некоторые операции над векторами и их результаты вычислений.

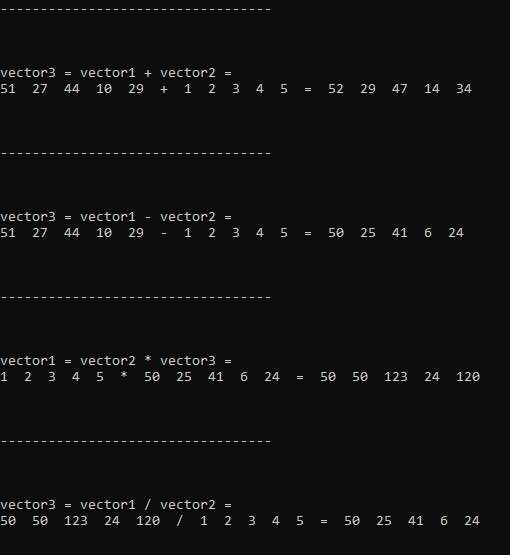


Рисунок 2. Операции над векторами.

После этого программа предлагает ввести 6 значений для того, чтобы заполнить матрицу размером 2х3.

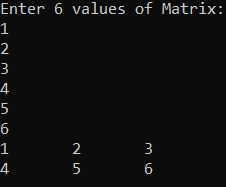


Рисунок 3. Пример заполнения матрицы

После заполнения матрицы, программа выведет на экран операции над матрицами с векторами.

Если ввод чисел был правильным, то программа будет завершена.

1. **Руководство программиста**

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

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

В решении Lab\_4 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 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. Метод изменения размера матрицы

| void Resize(int w, int l); |
| --- |

1. Вспомогательный метод получения количества вхождений элемента

| int GetIndex(); |
| --- |

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); |
| --- |

Внутри класса TVector определены следующие поля:

* T\* data – шаблонный указатель;
* int length – количество элементов вектора;

Внутри класса TVector определен слудующий набор public-методов:

1. Конструктор по умолчанию

| TVector(); |
| --- |

1. Конструктор инициализатор 1

| TVector(int c, T d); |
| --- |

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 l); |
| --- |

1. Метод получения длины вектора

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

1. Метод изменения длины вектора

| void Resize(int newLength); |
| --- |

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 = 0, int right = 0); |
| --- |

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); |
| --- |

* 1. ***Описание алгоритмов***

### 3.3.1. Алгоритм суммирования матриц:

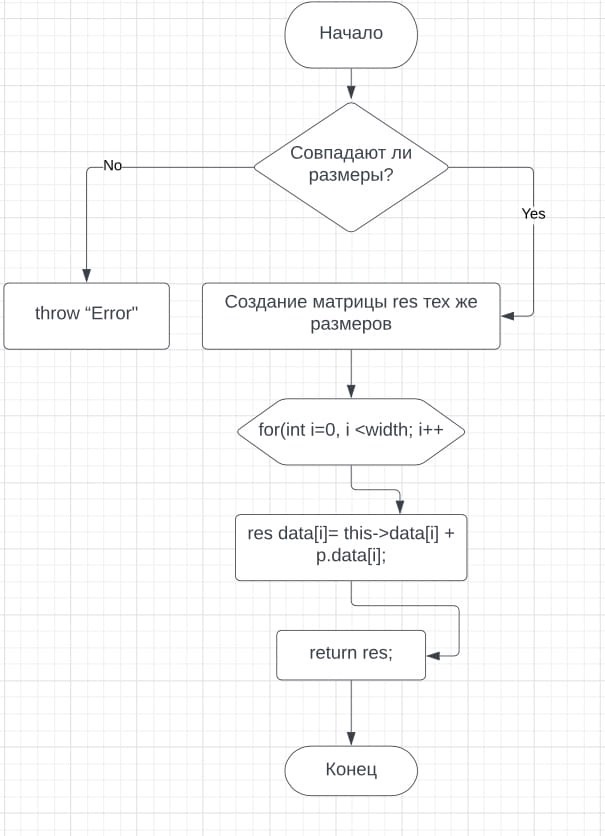


Рисунок 4. Сложение матриц блок-схема

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 | 0.20 |
| 2000 | 0.67 |
| 4000 | 2.50 |

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 | 7.1 |
| 2000 | 54.4 |
| 4000 | 602.3 |

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. Сортировка пузырьком – Режим доступа: [https://ru.wikipedia.org/wiki/Сортировка\_пузырьком](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC) - Загл. с экрана
3. Сортировка вставками – Режим доступа: [https://ru.wikipedia.org/wiki/Сортировка\_вставками](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0%D0%BC%D0%B8) - Загл. с экрана
4. Быстрая сортировка Хоара – Режим доступа: [https://ru.wikipedia.org/wiki/Быстрая\_сортировка](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) - Загл. с экрана

Объектно-ориентированное программирование – Режим доступа:

1. [https://ru.wikipedia.org/wiki/Объектно-ориентированное\_программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) - Загл. с экрана
2. Язык программирования С++ - Режим доступа:

<https://ru.wikipedia.org/wiki/C%2B%2B> – Загл. с экрана

1. **Приложения**

**7.1. Приложение 1 (main.cpp)**

#include "Vector.h"

#include "Matrix.h"

void whitespace()

{

std::cout << "\n\n\n----------------------------------\n\n\n\n";

}

int main()

{

TVector<int> vect1;

TVector<int> vect2(5, 2);

TVector<int> vect3(vector2);

vect1.Resize(5);

std::cin >> vect2;

for (int i = 0; i < vect1.GetLength(); i++)

vect1[i] = rand() % 50 + 10;

whitespace();

vect3 = vect1 + vect2;

std::cout << "vect3 = vect1 + vect2 =\n" << vect1 << "+ " << vect2 << "= " << vect3 << "\n";

whitespace();

vect3 = vect1 - vect2;

std::cout << "vect3 = vect1 - vect2 =\n" << vect1 << "- " << vect2 << "= " << vect3 << "\n";

whitespace();

vect1 = vect2 \* vect3;

std::cout << "vect1 = vect2 \* vect3 =\n" << vect2 << "\* " << vect3 << "= " << vect1 << "\n";

whitespace();

vect3 = vect1 / vect2;

std::cout << "vect3 = vect1 / vect2 =\n" << vect1 << "/ " << vect2 << "= " << vect3 << "\n";

whitespace();

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;

whitespace();

for (int i = 0; i < Matrix3.GetWidth(); i++)

for (int j = 0; j < Matrix3.GetLength(); j++)

Matrix3[i][j] = rand();

std::cout << "Matrix3 = \n" << Matrix3;

whitespace();

Matrix1 = Matrix2 + Matrix3;

std::cout << "Matrix1 = Matrix2 + Matrix3 =\n"

<< Matrix2 << "\n + \n\n" << Matrix3 << "\n = \n\n" << Matrix1 << "\n";

whitespace();

Matrix2 = Matrix3 - Matrix1;

std::cout << "Matrix2 = Matrix3 - Matrix1 =\n"

<< Matrix3 << "\n - \n\n" << Matrix1 << "\n = \n\n" << Matrix2 << "\n";

whitespace();

Matrix1.Resize(2, 2);

Matrix2.Resize(2, 2);

Matrix3.Resize(2, 2);

Matrix3 = Matrix2 \* Matrix1;

std::cout << "Matrix3 = Matrix2 \* Matrix1 =\n"

<< Matrix2 << "\n \* \n\n" << Matrix1 << "\n = \n\n" << Matrix3 << "\n";

whitespace();

for (int i = 0; i < Matrix4.GetWidth(); i++)

for (int j = 0; j < Matrix4.GetLength(); j++)

Matrix4[i][j] = rand() % 50;

vector1 = Matrix4 \* vector2;

std::cout << "vector1 = Matrix4 \* vector2 = \n" << Matrix4 << "\n \*\n\n" << vector2 << "\n\n =\n\n" << vector1;

whitespace();

vector1.QuickSort();

std::cout << "vector1 = " << vector1;

whitespace();

TMatrix<int> Matrix5(vector2.GetLength(), Matrix4.GetLength());

Matrix4.Resize(1, Matrix4.GetLength());

Matrix5 = vector2 \* Matrix4;

std::cout << "Matrix5 = vector2 \* Matrix4 = \n" << vector2 << "\n \*\n\n" << Matrix4

<< "\n\n = \n\n" << Matrix5;

whitespace();

return 0;

}

**7.2. Приложение 2 (vector.h)**

#pragma once

#include <iostream>

template <class T>

class TVector

{

public:

TVector();

TVector(int c, T d);

TVector(const TVector<T>& p);

~TVector();

void SetLen(int l);

int GetLength();

void Resize(int newLength);

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);

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\* data;

int length;

};

template<class T>

inline TVector<T>::TVector()

{

length = 0;

data = nullptr;

}

template<class T>

inline TVector<T>::TVector(int c, T d)

{

if (c > 0)

{

data = new T[c];

length = c;

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

data[i] = d;

}

else throw "Error";

}

template<class T>

inline TVector<T>::TVector(const TVector<T>& p)

{

if (p.data == nullptr)

{

data = nullptr;

length = 0;

}

else

{

length = p.length;

data = new T[length];

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

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

}

}

template<class T>

inline TVector<T>::~TVector()

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

length = 0;

}

template<class T>

inline void TVector<T>::SetLen(int l)

{

this->Resize(l);

}

template<class T>

inline int TVector<T>::GetLength()

{

return length;

}

template<class T>

inline void TVector<T>::Resize(int newLength)

{

if (newLength <= 0) throw "Error";

T\* mas = new T[newLength];

if (data != nullptr)

{

int index = 0;

if (length > newLength) index = newLength;

else index = length;

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

mas[i] = data[i];

delete[] data;

}

data = mas;

length = newLength;

}

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> A(length, 0);

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

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

return A;

}

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> A(length, 0);

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

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

return A;

}

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> A(length, 0);

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

A[i] = data[i] \* p.data[i];

return A;

}

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> A(length, 0);

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

if (p.data[i] == 0) A[i] = 0;

else A[i] = data[i] / p.data[i];

return A;

}

template<class T>

inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)

{

if (this == &p) return \*this;

if (data != nullptr) delete[] data;

if (p.data == nullptr)

{

length = 0;

data = nullptr;

}

else

{

length = p.length;

data = new T[length];

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

data[i] = p.data[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 (data[i] != p.data[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 (data[i] != p.data[i]) return true;

return false;

}

template<class T>

inline T& TVector<T>::operator[](int i)

{

if (data == nullptr)

throw "Error";

if (i < 0 || i >= length)

throw "Error";

return data[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 (data[j] < data[i])

{

int tmp = data[j];

data[j] = data[i];

data[i] = tmp;

}

}

template<class T>

inline void TVector<T>::InsertSort()

{

for (int i = 1; i < length; i++)

{

int k = i;

while (k > 0 && data[k - 1] > data[k])

{

int tmp = data[k - 1];

data[k - 1] = data[k];

data[k] = tmp;

k -= 1;

}

}

}

template<class T>

inline void TVector<T>::QuickSort(int left = 0, int right = 0)

{

int i, j;

if (right == 0)

{

left = 0;

right = length - 1;

}

i = left;

j = right;

int sr = data[(left + right) / 2];

int tmp;

while (i <= j) {

while (data[i] < sr) i++;

while (data[j] > sr) j--;

if (i <= j) {

tmp = data[i];

data[i] = data[j];

data[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)

{

for (int i = 0; i < A.GetLength(); i++)

B << A[i] << " ";

return B;

}

template<class T>

inline std::istream& operator>>(std::istream& B, TVector<T>& A)

{

std::cout << "Enter " << A.length << " values of vector: \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 l);

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->data = nullptr;

}

template<class T>

inline TMatrix<T>::TMatrix(int l) : TMatrix()

{

if (l < 0) throw "Error";

this->length = l;

width = l;

this->data = new TVector<T>[width];

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

this->data[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->data = new TVector<T>[width];

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

this->data[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.data == nullptr) this->data = nullptr;

else

{

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()

{

if (data != nullptr)

{

delete[] data;

data = 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.data[i] = this->data[i] + p.data[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.data[i] = this->data[i] - p.data[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->data[i][k] \* p.data[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->data[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->data != nullptr) delete[] data;

if (p.data == nullptr) data = nullptr;

else data = new TVector<T>[p.width];

for (int i = 0; i < p.width; i++)

this->data[i] = p.data[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->data[i][j] != this->p.data[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->data[i][j] != this->p.data[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->data[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>\* data1 = new TVector<T>[w];

if (this->data != 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++)

{

data1[i].Resize(l);

for (int j = 0; j < minL; j++)

data1[i][j] = this->data[i][j];

}

delete[] data;

}

this->data = data1;

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.data[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;

};