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. 3821Б1ПМ3

Нещеретов М.В.

Проверил:

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[**Введение** 3](#_Toc104153637)

[**1.** **Постановка задачи** 4](#_Toc104153638)

[**2.** **Руководство пользователя** 5](#_Toc104153639)

[**3.** **Руководство программиста** 6](#_Toc104153640)

[***3.1.*** ***Описание структуры программы*** 6](#_Toc104153641)

[***3.2.*** ***Описание структур данных*** 6](#_Toc104153642)

[***3.3.*** ***Описание алгоритмов*** 9](#_Toc104153643)

[3.3.1. Алгоритм суммирования матриц: 9](#_Toc104153644)

[3.3.2. Алгоритм умножения матриц: 10](#_Toc104153645)

[**4.** **Эксперименты** 11](#_Toc104153646)

[**5.** **Заключение** 11](#_Toc104153647)

[**6.** **Литература** 13](#_Toc104153648)

[**7.** **Приложения** 14](#_Toc104153649)

[**7.1. Приложение 1 (main.cpp)** 14](#_Toc104153650)

[}**7.2. Приложение 2 (vector.h)** 15](#_Toc104153651)

[**7.3. Приложение 3 (matrix.h)** 19](#_Toc104153652)

# **Введение**

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

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**

После запуска программы сразу создаётся 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.

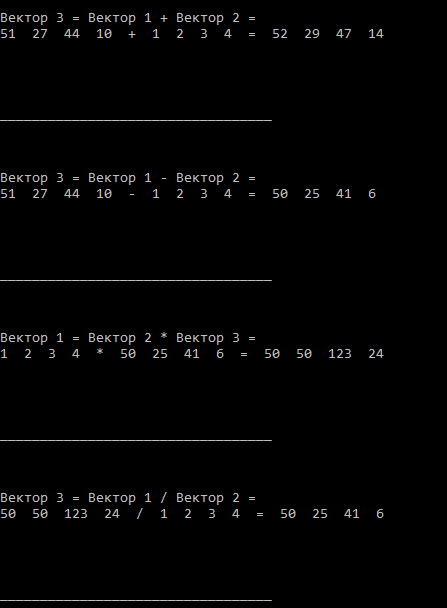


Рисунок 2. Пример операций над векторами

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

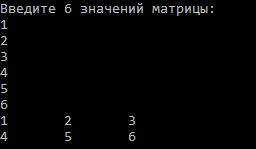


Рисунок 3. Пример заполнения матрицы

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 определено 3 модуля: main.cpp, Matrix.h, Vectors.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

Внутри класса TMatrix определен следующий набор public-методов:

1. Конструктор по умолчанию
2. Конструктор инициализатор 1
3. Конструктор инициализатор 2
4. Конструктор копирования
5. Деструктор
6. Перегрузка оператора +
7. Перегрузка оператора –
8. Перегрузка оператора \* (для матриц)
9. Перегрузка оператора \* (для умножения матрицы на вектор)
10. Перегрузка оператора =
11. Перегрузка оператора ==
12. Перегрузка оператора!=
13. Перегрузка оператора индексации
14. Метод получения количества строк матрицы
15. Метод задания количества строк матрицы
16. Метод изменения размера матрицы
17. Вспомогательный метод получения количества вхождений элемента
18. Перегрузка операции потокового вывода
19. Перегрузка операции потокового ввода

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

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
2. Конструктор инициализатор 1
3. Конструктор копирования
4. Деструктор
5. Перегрузка оператора +
6. Перегрузка оператора –
7. Перегрузка оператора \*
8. Перегрузка оператора /
9. Перегрузка оператора =
10. Перегрузка оператора ==
11. Перегрузка оператора!=
12. Перегрузка операции индексации
13. Метод задания длины вектора
14. Метод получения длины вектора
15. Метод изменения длины вектора
16. Метод, который позволяет отсортировать вектор пузырьком
17. Метод, который позволяет отсортировать вектор вставками
18. Метод, который позволяет отсортировать вектор сортировкой Хоара
19. Перегрузка оператора потокового вывода
20. Перегрузка оператора потокового ввода
    1. ***Описание алгоритмов***

### 3.3.1. Алгоритм суммирования матриц:

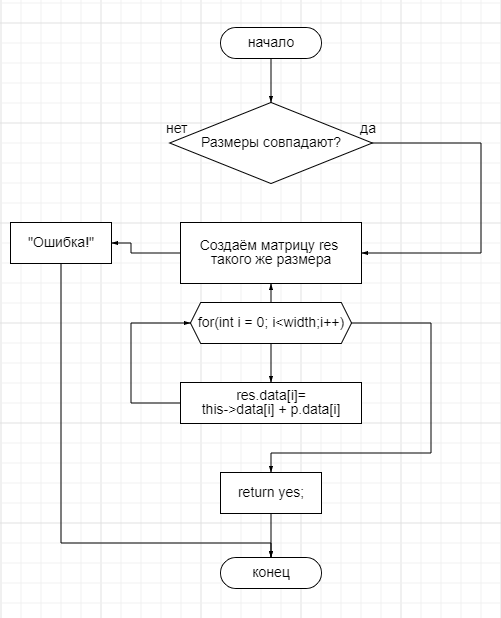


Рисунок 4. Сложение матриц блок-схема

### 3.3.2. Алгоритм умножения матриц:

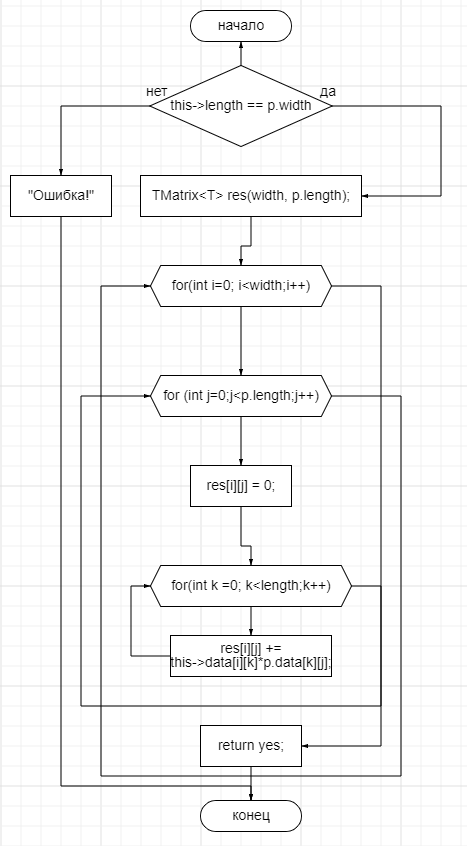


Рисунок 5. Умножение матриц блок-схема

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. **Заключение**

В ходе лабораторной работы была написана статическая библиотека С++, которая реализует вектор и матрицу. В этих классах реализованы основные математические операторы (+, -, \*, /).

В ходе выполнения данной работы я улучшил свои навыки написания классов, реализации функций классов и отработал подключение внешней статической библиотеки в языке C++.

Данная лабораторная работа помогла мне лучше понять суть шаблонов. Они действительно в разы упрощают написание кода программистам. Научился находить более эффективные методы реализации для тех или иных функций, да и в целом лучше понимать язык С++.

1. **Литература**

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

1. **Приложения**

**7.1. Приложение 1 (main.cpp)**

#include "Vectors.h"

#include "Matrix.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

TVector<int> v1;

TVector<int> v2(4, 2);

TVector<int> v3(v2);

v1.Resize(4);

cin >> v2;

for (int i = 0; i < v1.GetLength(); i++)

v1[i] = rand() % 50 + 10;

cout << "\n\n\n\n----------------------------------\n\n\n\n";

v3 = v1 + v2;

cout << "Вектор 3 = Вектор 1 + Вектор 2 =\n" << v1 << "+ " << v2 << "= " << v3 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

v3 = v1 - v2;

cout << "Вектор 3 = Вектор 1 - Вектор 2 =\n" << v1 << "- " << v2 << "= " << v3 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

v1 = v2 \* v3;

cout << "Вектор 1 = Вектор 2 \* Вектор 3 =\n" << v2 << "\* " << v3 << "= " << v1 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

v3 = v1 / v2;

cout << "Вектор 3 = Вектор 1 / Вектор 2 =\n" << v1 << "/ " << v2 << "= " << v3 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

TMatrix<int> m1;

TMatrix<int> m2(2, 3);

TMatrix<int> m3(m2);

TMatrix<int> m4(4, 5);

int\*\* indexs;

cin >> m2;

cout << m2;

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

for (int i = 0; i < m3.GetWidth(); i++)

for (int j = 0; j < m3.GetLength(); j++)

m3[i][j] = rand();

cout << "Матрица 3 = \n" << m3;

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

m1 = m2 + m3;

cout << "Матрица 1 = Матрица 2 + Матрица 3 =\n"

<< m2 << "\n + \n\n" << m3 << "\n = \n\n" << m1 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

m2 = m3 - m1;

cout << "Матрица 2 = Матрица 3 - Матрица 1 =\n"

<< m3 << "\n - \n\n" << m1 << "\n = \n\n" << m2 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

m1.Resize(2, 2);

m2.Resize(2, 2);

m3.Resize(2, 2);

m3 = m2 \* m1;

cout << "Матрица 3 = Матрица 2 \* Матрица 1 =\n"

<< m2 << "\n \* \n\n" << m1 << "\n = \n\n" << m3 << "\n";

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

for (int i = 0; i < m4.GetWidth(); i++)

for (int j = 0; j < m4.GetLength(); j++)

m4[i][j] = rand() % 50;

v1 = m4 \* v2;

cout << "Вектор 1 = Матрица 4 \* Вектор 2 = \n" << m4 << "\n \*\n\n" << v2 << "\n\n =\n\n" << v1;

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

TMatrix<int> m5(v2.GetLength(), m4.GetLength());

m4.Resize(1, m4.GetLength());

m5 = v2 \* m4;

cout << "Матрица 5 = Вектор 2 \* Матрица 4 = \n" << v2 << "\n \*\n\n" << m4

<< "\n\n = \n\n" << m5;

cout << "\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n\n";

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);

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 "Ошибка!";

}

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 "Ошибка!";

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 "Ошибка";

if (length != p.length) throw "Ошибка";

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 "Ошибка";

if (length != p.length) throw "Ошибка!";

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 "Ошибка!";

if (length != p.length) throw "Ошибка!";

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 "Ошибка!";

if (length != p.length) throw "Ошибка!";

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 "Ошибка!";

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

throw "Ошибка!";

return data[i];

}

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 << "Введите " << A.length << " значения вектора: \n";

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

B >> A[i];

return B;

}

**7.3. Приложение 3 (matrix.h)**

#pragma once

#include "Vectors.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);

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;

};

template<class T>

inline TMatrix<T>::TMatrix()

{

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 < this->length; i++)

this->data[i].Resize(this->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(this->length);

}

template<class T>

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

{

this->length = p.length;

width = p.width;

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 (this->data != nullptr)

{

delete[] this->data;

this->data = nullptr;

}

width = 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 < this->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[] this->data;

if (p.data == nullptr) this->data = nullptr;

else this->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[] this->data;

}

this->data = data1;

this->length = l;

width = w;

}

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 << "Введите " << A.width \* A.length << " значений матрицы: \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;

};