МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ 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

**Классы для работы с векторами и матрицами**

Выполнил:

студент ф-та ИТММ ПМИ – 381903-3

Стеценко Данил

Проверил:

ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020 г.

**Содержание**

[Введение 3](#_30j0zll)

[Постановка задачи 4](#_1fob9te)

[Руководство пользователя 5](#_3znysh7)

[Руководство программиста 6](#_2et92p0)-10

[Эксперименты 11-1](#_3dy6vkm)3

[Заключение 1](#_1t3h5sf)4

[Литература 1](#_4d34og8)5

[Приложение 16-2](#_2s8eyo1)9

# 1.Введение

В математике матрица - это прямоугольная таблица, состоящая из строк и столбцов, содержащих числа. Вектор- направленный отрезок прямой, то есть отрезок, для которого указано, какая из его граничных точек является началом, а какая — концом. Векторами называются матрицы, состоящие из одной строки или одного столбца

Для вектора определены следующие [алгебраические](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) операции:

* [сложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) векторов, имеющих один и тот же размер;
* вычитание векторов, имеющих один и тот же размер
* умножение векторов, имеющих один и тот же размер

умножение матрицы на элемент основного кольца или поля (то есть [скаляр](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80)).

Для матрицы определены следующие [алгебраические](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) операции

* [умножение, сложение матриц](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86) подходящего размера (матрицу, имеющую n nстолбцов, можно умножить справа на матрицу, имеющую n строк);
* в том числе умножение на матрицу вектора (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);
* умножение матрицы на элемент основного кольца или поля (то есть [скаляр](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80)).

# 2.Постановка задачи

1. Написать классы для работы с векторами и матрицами использовать шаблоны.

2. 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).
* в классе вектор должна быть возможность отсортировать его тремя способами.

3. Сравнить время работы, и сделать выводы.

# 3.Руководство пользователя

1. Создать объект типа vector или matrix выбирая между основными типами данных.
2. Произвести все необходимые операции над вектором или матрицей.
3. Чтобы использовать какую-нибудь функцию класса нужно обращаться к объекту класса через точку.

# 

# 4.Руководство программиста

***Описание структуры программы***

Программа состоит из одного решения, которое называется «vector».

В решении содержится 4 элементов: «vector.h», «header.h», «matrix.h», «main.cpp».

В «vector.h» определен класс MyVector**.**

В «matrix.h» определен класс Matric.

В «header.h» реализованы умножение вектора на матрицу и наоборот.

В «main.cpp» определена стандартная функция int main.

***Описание структуры программы***

В программе определены шаблонные классы MyVector и Matric.

Внутри класса Matric определены следующие поля(protected):

• M\*\* Matric – шаблонный двойной указатель

• int m – количество строк

• int n – количество столбцов

Внутри класса matrix определены следующие поля(public):

• Matrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля

• Matrix(int a, int b) – конструктор инциализатор, который устанавливает параметры размеров, но сам массив пустой

• Matrix(int с, int d, M\*\* arr) – конструктор инциализатор, который устанавливает размеры и массив

• Matrix(const Matrix[M]& other) – конструктор копирования, принимает на вход объект типа matrix, создает объект с теми же характеристиками, что и переданный, и матрицей того же содержимого

• ~Matrix() – деструктор

• int getM() – метод, возвращающий количество строк

• int getN() – метод, возвращающий количество столбцов

• M\*\* getMatric() – метод, возвращающий массив

• void setRazmernost(int,int) – метод, устанавливающий размерности матрицы

• void setMatric(M\*\* e) – метод, устанавливающий значение массива в матрицу

• friend Matriс operator +(const Matriс & other) - перегрузка оператора суммы

• friend Matriс operator –(const Matriс& other) - перегрузка оператора разности

• friend Matriс operator \*(const Matric& other) - перегрузка оператора умножения

• M\*& operator [](const int i) - перегрузка оператора индексации

• Matrix operator =(const Matric & other) - перегрузка оператора равенства

• bool operator ==(const Matric& other) - перегрузка оператора сравнения

• friend ostream& operator <<(ostream& out, const Matrix& mat) - перегрузка оператора вывода, позволяет выводить матрицы на экран

• friend istream& operator >>(istream& in, Matrix& mat) - перегрузка оператора ввода

Private поле:

• void MemorySize(int, int) - выделяет память до размеров матрицы, переданных в функцию;

Внутри класса MyVector определены следующие поля(protected):

• Vector\* massiv – шаблонный указатель(массив)

• int razmer – размерность вектора

Внутри класса MyVector определены следующие поля(public):

• MyVector() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля

• MyVector(int , Vector\*) – конструктор инциализатор, который устанавливает размер и массив

• MyVector(const MyVector[Vector]& other) – конструктор копирования, принимает на вход объект типа MyVector, создает объект с теми же характеристиками, что и переданный

• ~MyVector() – деструктор

• int getRazmer() – метод, возвращающий размерность вектора

• Vector\* getVector() – метод, возвращающий вектор(массив)

• void setRazmer(int) – метод, устанавливающий размерность вектора

• void setVector(Vector\* ) – метод, устанавливающий значение вектора

• MyVector operator +(const MyVector & other) - перегрузка оператора суммы

• MyVector operator –(const MyVector & other) - перегрузка оператора разности

• MyVector operator \*(const MyVector & other) - перегрузка оператора умножения

• MyVector operator =(const MyVector & other) - перегрузка оператора равенства

• bool operator ==(const MyVector & other) - перегрузка оператора сравнения

• friend ostream& operator <<(ostream& out, const MyVector & vec) - перегрузка оператора вывода, позволяет выводить матрицы на экран

• friend istream& operator >>(istream& in, MyVector & vec) - перегрузка оператора ввода

• Vector operator [](const int) - перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу

• MyVector operator /(MyVector& other) – перегрузка оператора / возвращает вектор, координаты которого являются частным от деления двух соответствующих координат операндов

• clock\_t BubbleSort() - сортировка «Пузырек».

• clock\_t InsertionSort() - сортировка «Вставка».

• clock\_t QuickSort() - сортировка «Быстрая».

***Описание алгоритмов***

1. **Сортировка пузырьком**

Главной идеей алгоритма является то, что сравниваются два соседних элемента, и если они стоят в неправильном порядке, то эти элементы меняются местами. Таким образом, за каждый проход в конце массива встает наибольшее число, если сортируется по возрастанию, или наименьшее, если по убыванию. Благодаря этому, программа может не проверят его на следующей итерации.

1. **Сортировка вставкой**

На вход алгоритма подаётся последовательность n чисел: a1, а2, ..., аn . Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a’1, а’2, …, а’n , чтобы выполнялось следующее соотношение a’1 ≤ а’2 ≤ … ≤а’n .

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

1. **Быстрая сортировка**

Общая идея алгоритма состоит в следующем:

• Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

• Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

• Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

# 

# 5.Эксперименты

1. Суммирование матриц:

Matrix<M> F (m, n, other.massiv);

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

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

F.massiv[i][j] = other.massiv[i][j] + massiv[i][j];

return F;

Асимптотическая сложность:

Замеры времени сложения матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.62 |
| 2000 | 2.15 |
| 4000 | 8.42 |

Таблица 1: Время суммирования матриц.

Вывод: при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

1. Умножение матриц:

for (int i = 0; i < mat.m; i++)

for (int j = 0; j < mat.n; j++){

mat.massiv[i][j] = 0;

for (int k = 0; k < n; k++){

mat.massiv[i][j] = mat.massiv[i][j] + (massiv[i][k] \* other.massiv[k][j]);

}

}

Асимптотическая сложность:

Замеры времени умножения матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 9.40 |
| 2000 | 79.37 |
| 4000 | 6034.6 |

Таблица 2: Время умножения матриц.

Вывод: При увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 8 раза.

1. Сложность векторно-матричного умножения:

T\*\*arr=new T\*[matrix.GetN()];

for (int i = 0; i < vector.GetSize(); i++)

for (int j = 0; j < matrix.GetN(); j++){

arr[i][j]=(matrix[0][j]\*vector[i];

}

Асимптотическая сложность:

Замеры времени умножения матриц и векторов:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.002 |
| 2000 | 0.014 |
| 4000 | 0.053 |

Таблица 3: Время умножения матриц и векторов

Вывод: При увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

Замеры времени работы сортировок:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сортировка пузырьком | Сортировка вставками | Быстрая сортировка |
| 100 000 элементов | 14.98 сек | 1.20 сек | 0.013 сек |
| 200 000 элементов | 62.16 сек | 7.28 сек | 0.025 сек |
| 400 000 элементов | 253.30 сек | 29.20 сек | 0.057 сек |
| Асимптотическая сложность |  |  |  |

Таблица 4: Время работы сортировок

Вывод: Затрачиваемое на реализацию операции время растет эквивалентно асимптотической функции.

# 6.Заключение

Мной была написана программа, позволяющая работать с векторами и матрицами. В частности выполнять все алгебраические операции с использованием шаблонов.

# 

# 7.Литература

1. Бьерн Страуструп. Язык программирования C++, 2-е изд: Пер. с англ. — М.: Издательский дом «Бином», 2017. — 1136 с.
2. Герберт Шилдт. C++ для начинающих. Шаг за шагом, 2-е изд: Пер. с англ – М.: Издательский дом «ЭКОМ Паблишерз», 2013. – 640 с.
3. Липпман Стенли Б., Лажойе Жози, Му Барбара Э. Язык программирования C++, базовый курс, 5-е изд: Пер. с англ. – М.: Издательский дом «Диалектика», 2018. – 1118 с.

# 

# 8.Приложение

***Приложение 1. Код программы***

**«vector.h»**

#pragma once

#include <iostream>

#include <clocale>

#include <iomanip>

#include"Matrix.h"

using namespace std;

template <typename Vector>

class MyVector{

protected:

int size;

Vector\* array;

public:

MyVector();

MyVector(int r, Vector\* A);

MyVector(MyVector& B);

~MyVector();

Vector\* GetVector();

int Getsize();

void SetVector(Vector\* C);

void Setsize(int e);

MyVector<Vector> operator=(const MyVector<Vector>& other);

bool operator ==(const MyVector<Vector>& other);

Vector operator [](const int i);

MyVector<Vector> operator + (const MyVector& other);

MyVector<Vector> operator-(const MyVector<Vector>& other);

Vector operator\*(const MyVector<Vector>& other);

MyVector<Vector> operator /(MyVector<Vector>& other);

template <typename Vector>

friend ostream& operator<<(ostream& os, const MyVector<Vector>& vec);

template <typename Vector>

friend istream& operator>>(istream& in, MyVector<Vector>& vec);

clock\_t BubbleSort();

clock\_t InsertionSort();

clock\_t QuickSort(int first, int last);

int search(Vector a);

};

template<typename Vector>

MyVector<Vector>::MyVector(int r,Vector\*A){

array = NULL;

try{

if (r > 0){

size = r;

array = new Vector[size];

for (int i = 0; i < size; i++){

array[i] = A[i];

}

}

else throw 1;

}

catch (int i){

cout << "oshibka vvoda dannux, array=0" << "\n";

exit(0);

}

}

template<typename Vector>

MyVector<Vector>::MyVector(MyVector& B){

array = nullptr;

size = B.Getsize();

array = new Vector[size];

for (int i = 0; i < size; i++){

array[i] = B.array[i];

}

}

template<typename Vector>

Vector\* MyVector<Vector>::GetVector(){

return array;

}

template<typename Vector>

int MyVector<Vector>::Getsize(){

return size;

}

template<typename Vector>

void MyVector<Vector>::SetVector(Vector\* C){

int i = strlen(C);

if (array != 0){

array = 0;

delete[] array;

}

size = i;

array = new Vector[size];

for (int k = 0; k < i; k++){

array[k] = C[k];

}

}

template<typename Vector>

MyVector<Vector>::MyVector(){

size = 1;

array = new Vector[1];

Vector a = 0;

array[0] = a;

}

template<typename Vector>

MyVector<Vector>::~MyVector(){

if (array != 0){

array = 0;

delete[] array;

}

size = 0;

}

template<typename Vector>

MyVector<Vector> MyVector<Vector>::operator=(const MyVector<Vector>& other){

if (\*this == other)

return \*this;

delete[] array;

this->size = other.size;

array = new Vector[size];

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

array[i] = other.array[i];

return \*this;

}

template<typename Vector>

bool MyVector<Vector>::operator==(const MyVector<Vector>& other){

int k = 0;

if (size == other.size){

for (int i = 0; i < size; i++){

if (array[i] == other.array[i]){

k = k + 1;

}

}

if (k == size){

return true;

}

else return false;

}

else

return false;

}

template<typename Vector>

Vector MyVector<Vector>::operator[](const int i){

return array[i];

}

template<typename Vector>

MyVector<Vector> MyVector<Vector>::operator + (const MyVector& other){

if (size == other.size){

MyVector<Vector> vector(size, other.array);

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

vector.array[i] = array[i]+other.array[i];

return vector;

}

}

template<typename Vector>

void MyVector<Vector>::Setsize(int e){

size = e;

}

template<typename Vector>

MyVector<Vector> MyVector<Vector>::operator-(const MyVector<Vector>& other){

if (size == other.size){

MyVector<Vector> vector(size, other.array);

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

vector.array[i] = array[i]-other.array[i];

return vector;

}

else

throw - 1;

}

template<typename Vector>

Vector MyVector<Vector>::operator \*(const MyVector<Vector>& other){

Vector res = 0;

if (size == other.size){

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

res = res + (array[i] \* other.array[i]);

return res;

}

}

template<typename Vector>

MyVector<Vector> MyVector<Vector>::operator /(MyVector<Vector>& other){

MyVector<Vector> K;

K = \*this;

for (int i = 0; i < size; i++){

if (K.array[i] != 0){

K.array[i] = K.array[i] / other.array[i];

}

}

return K;

}

template<typename Vector>

ostream& operator<<(ostream& os, const MyVector<Vector>& vec){

os << "(";

for (int i = 0; i < vec.size; i++){

os << vec.array[i];

if (i < vec.size - 1)

os << " ";

}

os << ")^T";

return os;

}

template<typename Vector>

istream& operator>>(istream& in, MyVector<Vector>& vec){

cout << "Enter size: ";

in >> vec.size;

Vector\* mas;

mas=new Vector[vec.size];

for (int i = 0; i < vec.size; i++){

cout << "[" << i << "]" << "\n";

in >> mas[i];

}

MyVector<Vector> V(vec.size,mas);

vec = V;

mas = 0;

delete[] mas;

return in;

}

template<typename Vector>

clock\_t MyVector<Vector>::BubbleSort(){

clock\_t start, finish;

start = clock();

Vector tmp = 0;

int i, j = 0;

start = clock();

for (int i = 0; i < size; i++){

for (int j = 0; j < i; j++){

if (array[j] > array[j + 1]){

tmp = array[j];

array[j] = array[j + 1];

array[j + 1] = tmp;

}

}

}

finish = clock();

return (finish - start);

}

template<typename Vector>

clock\_t MyVector<Vector>::InsertionSort(){

int k = 0;

clock\_t start, finish;

start = clock();

Vector tmp;

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

{

tmp = array[i];

k = i - 1;

while (k >= 0 && array[k] > tmp)

{

array[k + 1] = array[k];

k = k - 1;

}

array[k + 1] = tmp;

}

finish = clock();

return (finish - start);

}

template<typename Vector>

clock\_t MyVector<Vector>::QuickSort(int first, int last){

clock\_t t1, t2, t3;

t1 = clock();

int i = first, j = last;

Vector tmp;

Vector x = array[(first + last) / 2];

do{

while (array[i] < x)

i++;

while (array[j] > x)

j--;

if (i <= j){

if (i < j){

tmp = array[i];

array[i] = array[j];

array[j] = tmp;

}

i++;

j--;

}

} while (i <= j);

if (i < last)

t3 = QuickSort(i, last);

if (first < j)

t3 = QuickSort(first, j);

t2 = clock();

return t2 - t1;

}

template<typename Vector>

int MyVector<Vector>::search(Vector a){

int quantity = 0;

for (int i = 0; i < size; i++){

if (array[i] == a){

quantity = quantity + 1;

}

}

return quantity;

}

**«matrix.h»**

#pragma once

#include<iostream>

using namespace std;

template<typename M>

class Matrix{

private:

void MemorySize(int width, int length){

array = new M \* [m];

for (int i = 0; i < width; i++) this->array[i] = new M[length];

}

protected:

int m;

int n;

M\*\* array;

public:

Matrix();

Matrix(int a, int b);

Matrix(int c, int d, M\*\* arr);

Matrix(const Matrix<M>& other);

~Matrix();

M\*\* GetMatrix();

int GetM();

int GetN();

void SetSize(int width, int length);

void SetMatrix(M\*\* e);

Matrix<M> operator=(const Matrix<M>& other);

bool operator ==(const Matrix<M>& other);

M\*& operator [](const int i);

Matrix<M> operator+(const Matrix<M>& other);

Matrix<M> operator-(const Matrix<M>& other);

Matrix<M> operator\*(const Matrix<M>& other);

template<typename M>

friend ostream& operator<<(ostream& os, const Matrix<M>& mat);

template<typename M>

friend istream& operator>>(istream& in, Matrix<M>& mat);

int search(M a);

};

template<typename M>

Matrix<M>::Matrix(){

m = 1;

n = 1;

MemorySize(m, n);

array[0][0] = 0;

}

template<typename M>

Matrix<M>::Matrix(int a,int b){

m = a;

n = b;

MemorySize(m, n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

array[i][j] = 0;

}

}

}

template<typename M>

Matrix<M>::Matrix(int c, int d, M\*\* arr){

m = c;

n = d;

MemorySize(c, d);

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

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

array[i][j] = arr[i][j];

}

template<typename M>

Matrix<M>::Matrix(const Matrix<M>& other){

m = other.n;

n = other.m;

MemorySize(m, n);

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

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

array[i][j] = other.array[i][j];

}

template<typename M>

Matrix<M>::~Matrix(){

if (array != 0){

for (int i = 0; i < m; i++){

array[i] = 0;

delete[] array[i];

}

array = 0;

delete[] array;

n = 0;

m = 0;

}

}

template<typename M>

int Matrix<M>::search(M a){

int quantity = 0;

for (int i = 0; i < m; i++){

for (int j = 0; j < n; j++){

if (array[i][j] == a){

quantity = quantity + 1;

}

}

}

return quantity;

}

template<typename M>

int Matrix<M>::GetM(){

return m;

}

template<typename M>

int Matrix<M>::GetN(){

return n;

}

template<typename M>

M\*\* Matrix<M>::GetMatrix(){

return array;

}

template<typename M>

void Matrix<M>::SetSize(int width, int length){

m = width;

n = length;

}

template<typename M>

void Matrix<M>::SetMatrix(M\*\* e){

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

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

array[i][j] = e[i][j];

}

template<typename M>

Matrix<M> Matrix<M>::operator=(const Matrix<M>& other){

if (\*this == other)

return \*this;

for (int i = 0; i < m; i++){

array[i] = 0;

delete[] array[i];

}

delete[] array;

array = 0;

m = other.m;

n = other.n;

MemorySize(m, n);

for (int i = 0; i < m; i++){

for (int j = 0; j < n; j++){

array[i][j] = other.array[i][j];

}

}

return \*this;

}

template<typename M>

M\*& Matrix<M>::operator [](const int i){

return array[i];

}

template<typename M>

bool Matrix<M>::operator==(const Matrix<M>& other){

if (m == other.m && n == other.n){

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

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

if (array[i][j] != other.array[i][j])

return false;

return true;

}

else

return false;

}

template<typename M>

Matrix<M> Matrix<M>::operator+(const Matrix<M>& other){

if (n == other.n && m == other.m){

Matrix<M> F(m,n,other.array);

for (int i = 0; i < m; i++){

for (int j = 0; j < n; j++){

F.array[i][j] = other.array[i][j] + array[i][j];

}

}

return F;

}

}

template<typename M>

Matrix<M> Matrix<M>::operator-(const Matrix<M>& other){

if (n == other.n && m == other.m){

Matrix<M> F(m, n);

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

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

F.array[i][j] = array[i][j]-other.array[i][j];

return F;

}

else

throw - 1;

}

template<typename M>

Matrix<M> Matrix<M>::operator\*(const Matrix<M>& other){

if (n == other.m){

Matrix<M> mat(m,(other.n));

for (int i = 0; i < mat.m; i++)

for (int j = 0; j < mat.n; j++){

mat.array[i][j] = 0;

for (int k = 0; k < n; k++){

mat.array[i][j] = mat.array[i][j] + (array[i][k] \* other.array[k][j]);

}

}

return mat;

}

else

throw - 1;

}

template<typename M>

ostream& operator<<(ostream& os, const Matrix<M>& mat){

cout << "\n";

for (int i = 0; i < mat.m; i++){

if (i > 0) os << endl;

for (int j = 0; j < mat.n; j++)

os << mat.array[i][j] << " ";

}

return os;

}

template<typename M>

istream& operator>>(istream& in, Matrix<M>& mat){

if ((mat.GetM()!=0)||(mat.GetN()!=0)){

for (int i = 0; i < mat.m; i++)

delete[] mat.array[i];

delete[] mat.array;

}

cout << "Enter size" << "\n";

cout << "m = ";

in >> mat.m;

cout << "n = ";

in >> mat.n;

mat.MemorySize(mat.m,mat.n);

for (int i = 0; i < mat.m; i++){

cout << "Enter " << i + 1 << " line" << "\n";

for (int j = 0; j < mat.n; j++)

in >> mat.array[i][j];

}

return in;

}

**«header.h»**

#pragma once

#include "Vector.h"

#include "Matrix.h"

template <typename T>

MyVector<T> operator \*(Matrix<T>& matrix, MyVector<T>& vector){

if (matrix.GetM() == vector.Getsize()){

T\* arr = new T[matrix.GetM()];

T\*\* arr\_matrix = matrix.GetMatrix();

T\* arr\_vector = vector.GetVector();

for (int i = 0; i < matrix.GetM(); i++){

arr[i]=0;

for (int j = 0; j < matrix.GetN(); j++){

arr[i]=arr[i] + (arr\_matrix[i][j] \* arr\_vector[j]);

}

}

MyVector<T> result(matrix.GetN(), arr);

delete[] arr;

return result;

}

};

template <typename T>

Matrix<T> operator \*(MyVector<T>& vector, Matrix<T>& matrix){

if (matrix.GetN() == vector.Getsize() && matrix.GetM() == 1){

T\*\* arr = new T \* [matrix.GetN()];

for (int i = 0; i < vector.Getsize(); i++)

arr[i] = new T[vector.Getsize()];

for (int i = 0; i < vector.Getsize(); i++)

for (int j = 0; j < matrix.GetN(); j++){

arr[i][j]= (matrix[0][j] \* vector[i]);

}

Matrix<T> result(matrix.GetN(), vector.Getsize(), arr);

for (int i = 0; i < vector.Getsize(); i++)

delete[] arr[i];

delete[] arr;

return result;

}

};

**«main.cpp»**

#include"Matrix.h"

#include <cstdlib>

#include"Vector.h"

#include <time.h>

#include <clocale>

#include <malloc.h>

#include <vector>

#include<iostream>

#include <iomanip>

#include"Header.h"

using namespace std;

int main(){

MyVector<int> A;

cin >> A;

cout << "your vector:";

cout << A<<endl;

int K = 0;

int arr[] = { 7,5,1,4,2,6,9 };

int arr1[] = { 2,7,8,5,6,3,4 };

MyVector<int> B(7, arr);

MyVector<int> C(7, arr1);

MyVector<int> D;

cout << "+ : ";

D = B + C;

cout << D<<endl;

cout << "\* : ";

K = B \* C;

cout << K << endl;

cout << "- : ";

D = B - C;

cout << D << endl;

cout << "/ : ";

D = C / B;

cout << D << endl;

cout << "--------------------" << endl;

cout << "Matrix 1" << "\n";

Matrix<int> AB;

cin >> AB;

cout << "square matrix 1:" << AB << endl;

cout << "enter the same MatrixSize " << endl;

Matrix<int> AC;

cin >> AC;

cout << "square matrix 2:" << AB << endl<<endl;

Matrix<int> ABC;

cout << "+ : ";

ABC = AB + AC;

cout << ABC << endl << endl;

cout << "\* : ";

ABC = AB \* AC;

cout << ABC << endl << endl;

cout << "- : ";

ABC = AB - AC;

cout << ABC << endl << endl;

return 0;

}