Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н. И. Лобачевского»

(ННГУ)

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Векторы и Матрицы

Выполнил:

Студент ИИТММ

Гр. 381906-2

Челебаев Артём

Проверил:

Лебедев И. Г.

Нижний Новгород

2020

**Содержание**

[Введение](#_Введение)

1. [Постановка задачи](#_Постановка_задачи)
2. [Руководство пользователя](#_Руководство_пользователя)
3. [Руководство программиста](#_Руководство_программиста)
   1. [Описание структуры программы](#_Описание_структуры_программы)
   2. [Описание структур данных](#_Описание_структур_данных)
   3. [Описание алгоритмов](#_Описание_алгоритмов)
4. [Эксперименты](#_Эксперименты)
5. [Заключение](#_Заключение)
6. [Литература](#_Литература)
7. [Приложение](#_Приложение)

# **Введение**

Матричная алгебра - раздел алгебры, посвященный правилам действий над матрицами - одним из самых важных, употребительных и содержательных понятий в математике.

Матрицы проникли почти во все отрасли человеческой деятельности. В математике они используются при исследовании систем m линейных уравнений с n неизвестными. В экономике - при отражении соотношений затрат, производственных и экономических структур. В технике - при расчете сооружений. В физике матрицы применены для повышения точности вычисления значений полей вблизи неоднородности, теории управления, статистики, других областей науки и знаний.

В данной работе 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:

* [сложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)/вычитание матриц/векторов, имеющих один и тот же размер;
* скалярное произведение векторов, имеющих один и тот же размер
* [умножение 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 столбцов, можно умножить справа на матрицу, имеющую  n строк);
* в том числе умножение вектора на матрицу и наоборот (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);

# **Постановка задачи**

Необходимо написать классы для работы с матрицами и векторами, при этом использовать шаблоны.

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);

Также в классе вектор должна быть возможность отсортировать его тремя способами (пузырек, вставка, быстрая сортировки). Также нужно сравнить время работы, сделать выводы.

# **Руководство пользователя**

Для начала программа выводит пользователю консольный интерфейс, где предлагается выбрать тип данных, с которым предстоит оперировать;

Далее пользователь выбирает желаемую операцию и затем вводит необходимые данные.

# **Руководство программиста**

# **Описание структуры программы**

Четыре файла: Matrix.h, Vector.h, Operations.h, main.cpp. Первые два содержат классы матриц и векторов соответственно, а также описания их методов. Третий файл содержит описания матрично-векторных операций. Последний файл для работы со всеми данными программы.

# **Описание структур данных**

В программе содержатся следующие шаблонные классы:

Class Matrix

Class Vector

Внутри класса Matrix определены следующие поля:

* T\*\* arr – шаблонный двойной указатель;
* int n – количество строк матрицы;
* int m– количество столбцов матрицы;

Внутри класса Matrix определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Matrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0, через списки инициализации;
* Matrix(int size1, int size2, T\*\* arr\_) – конструктор инциализатор, принимает на вход два параметра типа int и один шаблонный двойной указатель, создает динамический массив заданных размеров, заполняет его теми же элементами какими заполнен переданный указатель, поля n и m, становятся значениями переданными в конструктор соответственно;
* Matrix(int size1, int size2) ) - конструктор инциализатор, принимает на вход два параметра типа int, создает динамический массив заданного размера, заполняет его случайными числами.
* Matrix(const Matrix<T> & 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, очищает выделенную методами и конструкторами память;
* int GetN() – возвращает количество строк матрицы;
* int GetM() – возвращает количество столбцов матрицы;
* void Set(int \_n, int \_m, T\*\* \_arr) – принимает на вход два параметра типа int и один шаблонный двойной указатель, создает динамический массив заданных размеров, заполняет его теми же элементами какими заполнен переданный указатель, поля n и m, становятся значениями переданными в конструктор соответственно;
* void SetI(int index1, int index2, T t) – принимает на вход два параметра типа int и одно шаблонное значение, находит в матрице находит элемент в index1 строке и index2 столбце и заменяет его на t;
* Метод Search1, в качестве аргумента принимающий число и затем осуществляющий поиск количества вхождений этого числа в матрице. Перегружен ещё два раза для работы с типами данных float и double помимо int. Аналогичный метод присутствует и в классе Vector.
* Метод Search2, в качестве аргумента принимающий число и затем осуществляющий поиск всех вхождений этого числа в матрице. Перегружен ещё два раза для работы с типами данных float и double помимо int. Аналогичный метод присутствует и в классе Vector.
* T\*& operator [](const int \_n) – перегрузка оператора индексации, позволяет возвращать элемент матрицы по индексу;
* Matrix<T> operator+(const Matrix<T>& other) – перегрузка оператора суммы, позволяет возвращать матрицу, являющуюся суммой двух матриц;
* Matrix<T> operator-(const Matrix<T>& other) – перегрузка оператора разности, позволяет возвращать матрицу, являющуюся разностью двух матриц;
* Matrix<T> operator = (const Matrix<T>& other) – перегрузка оператора присвоить, позволяет присваивать один объект типа Matrix другому;
* bool operator ==(const Matrix<T>& other) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Matrix;
* Matrix<T> operator\*(const Matrix<T>& other) – перегрузка оператора умножить, позволяет перемножать обьекты типа Matrix;
* friend ostream& operator<<(ostream& os, const Matrix<T>& mat) – перегрузка оператора вывода, позволяет выводить матрицы на экран;
* friend istream& operator>>(istream& in, Matrix<T>& mat) – перегрузка оператора ввода, позволяет вводить матрицы.

Внутри класса Vector определены следующие поля:

* T\* arr – шаблонный указатель;
* int N – количество элементов вектора;

Внутри класса Vector определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Vector() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0, через списки инициализации;
* Vector(int size, const T\* \_arr) – конструктор инциализатор, принимает на вход параметр типа int и шаблонный указатель, создает динамический массив заданного размера, заполняет его теми же элементами какими заполнен переданный указатель;
* Vector(int size) - конструктор инциализатор, принимает на вход параметр типа int, создает динамический массив заданного размера, заполняет его случайными числами.
* Vector(const Vector<T>& other) – конструктор копирования, принимает на вход объект типа Vector, создает объект с теми же характеристиками что и переданный;
* ~Vector() – деструктор, очищает выделенную методами и конструкторами память;
* int GetN() – возвращает количество элементов вектора;
* void Set(int n, T\* \_arr) – принимает на вход параметр типа int и шаблонный указатель, создает динамический массив заданного размера, заполняет его теми же элементами какими заполнен переданный указатель;
* void SetI(int index, T t) – принимает на вход параметр типа int и шаблонное значение, находит в векторе находит элемент c индексом index и заменяет его на t;
* T operator [](const int i) – перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу;
* Vector<T> operator+(const Vector<T>& other) – перегрузка оператора суммы, позволяет возвращать вектор, являющийся суммой двух векторов;
* Vector<T> operator-(const Vector<T>& other) – перегрузка оператора разности, позволяет возвращать вектор, являющийся разностью двух векторов;
* Vector<T> operator = (const Vector<T>& other) – перегрузка оператора присвоить, позволяет присваивать один объект типа Vector другому;
* bool operator ==(const Vector<T>& other) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Vector;
* T operator\*(const Vector<T>& other) – перегрузка оператора умножить, позволяет перемножать объекты типа Vector;
* friend ostream& operator<<(ostream& os, const Vector<T>& vec)– перегрузка оператора вывода, позволяет выводить вектора на экран;
* friend istream& operator>>(istream& in, Vector<T>& vec) – перегрузка оператора ввода, позволяет вводить вектора;
* void Bubble\_sort() - сортировка методом «Пузырек»;
* void Insertion\_sort() - сортировка методом «Вставка»;
* void Quick\_sort(int first, int last) - сортировка методом «Хоара».

Также в модуле Operator.h реализованы две шаблонные функции:

* Vector<T> operator \*(Matrix<T>& matrix, Vector<T>& vector) – перегрузка оператора умножить, позволяет перемножать объекты типа Matrix с объектами типа Vector;
* Matrix<T> operator \*(Vector<T>& vector, Matrix<T>& matrix) – перегрузка оператора умножить, позволяет перемножать объекты типа Vector с объектами типа Matrix;

# **Описание алгоритмов**

1. Сортировка Пузырьком

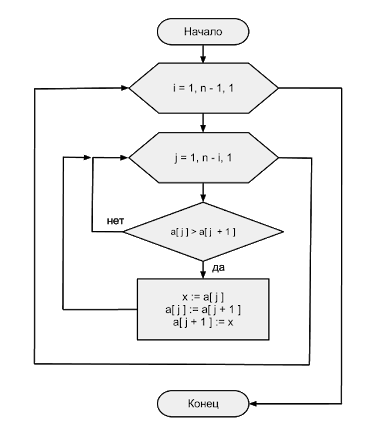


Рисунок 1. Блок схема сортировки Пузырьком

1. Сортировка Вставками

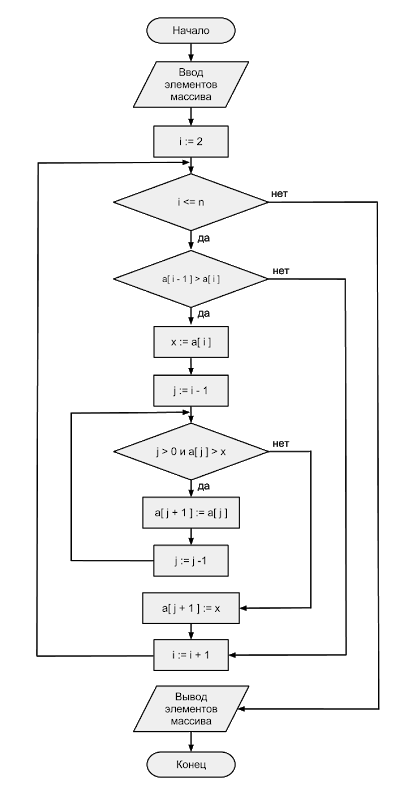


Рисунок 2. Блок-схема сортировки Вставками

3. Быстрая сортировка

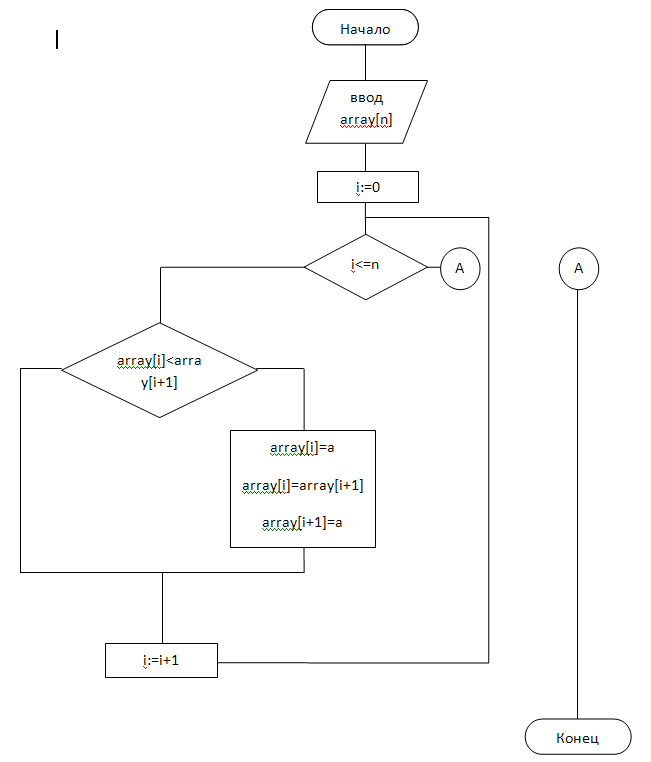


Рисунок 3. Блок-схема Быстрой сортировки

# **Эксперименты**

Оценим время, которое занимают матричные, векторно-матричные операции, с помощью асимптотической сложности.

1. Рассмотрим код, отвечающий за суммирование матриц:

Matrix<T> matrix (n, m, other.arr);

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

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

matrix.arr[i][j] = other.arr[i][j] + arr[i][j];

return matrix;

Асимптотическая сложность этого кода:



Теперь произведем замеры времени сложения матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.21 |
| 2000 | 0.81 |
| 4000 | 2.84 |

Таблица 1: Время суммирования матриц.

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

1. Теперь аналогично оценим асимптотическую сложность матричного умножения:

for (int i = 0; i < result.n; i++)

for (int j = 0; j < result.m; j++)

{

result.arr[i][j] = 0;

for (int k = 0; k < this->m; k++)

{

result.arr[i][j] = result.arr[i][j] + (arr[i][k] \* other.arr[k][j]);

}

}

Асимптотическая сложность умножения:



Теперь произведем замеры времени умножения матриц

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 6.574 |
| 2000 | 76.914 |
| 4000 | 854.447 |

Таблица 2: Время умножения матриц.

Как мы можем видеть, действительно, при увеличении количества элементов в 2 раза относительно предыдущего время увеличивается примерно 8 раза.

1. Теперь аналогично оценим асимптотическую сложность векторно-матричного умножения:

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

for (int i = 0; i < result.GetN(); i++)

for (int j = 0; j < result.GetM(); j++)

{

result.SetI(i, j, matrix[0][j] \* vector[i]);

}

Асимптотическая сложность этого кода :



Теперь мы произведем замеры времени умножения матриц и векторов

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.002 |
| 2000 | 0.01 |
| 4000 | 0.044 |

Таблица 3: Время умножения матриц и векторов

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

Теперь перейдем к сортировкам. Асимптотическая сложность уже определена. Поэтому просто произведём замеры и составим таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сортировка пузырьком | Сортировка вставками | Быстрая сортировка  (Хоара) |
| 10 000 элементов | 0.215 сек | 0.095 сек | 0.002 сек |
| 100 000 элементов | 22.193 сек | 7.218 сек | 0.019 сек |
| 250 000 элементов | 140.287 сек | 45.385 сек | 0.033 сек |
| Асимптотическая сложность |  |  |  |

Таблица 4: Время работы сортировок

Как видно из таблицы, действительно, затрачиваемое на реализацию операции время растет эквивалентно асимптотической функции, что подтверждает наши ожидания.

# **Заключение**

В данной программе получилось реализовать работу матричных, векторных и матрично-векторных операций. Помимо этого, наблюдения показали, что прогнозируемое время работы алгоритмов сортировок совпало с реальным.

# **Литература**

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. Вирт Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — С. 406.
3. Официальный сайт программы PMBK. – Режим доступа <http://www.pmbk.ru/pr/chto-takoe-massivy-v-programmirovanii.html>
4. Официальный сайт Bestreferat. – Режим доступа <https://www.bestreferat.ru/referat-183091.html>
5. Официальный сайт Habr. – Режим доступа <https://habr.com/ru/post/339656/>
6. Официальный сайт Microsoft. – Режим доступа <https://docs.microsoft.com/ru-ru/cpp/c-language/main-function-and-program-execution?view=vs-2019>

# **Приложение**

***Код программы***

**Matrix.h**

#pragma once

#include <iostream>

#include "Vector.h"

using namespace std;

template <class T>

class Matrix

{

protected:

int n;

int m;

T\*\* arr;

public:

Matrix();

Matrix(int size1, int size2, T\*\* arr\_);

Matrix(int size1, int size2);

Matrix(const Matrix<T>& other);

~Matrix();

int GetN();

int GetM();

T\*\* GetArr();

void Set(int \_n, int \_m, T\*\* \_arr);

void SetI(int index1, int index2, T t);

int Search1(int search);

float Search1(float search);

double Search1(double search);

void Search2(int search);

void Search2(float search);

void Search2(double search);

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

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

T\*& operator [](const int \_n);

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

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

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

template<class T>

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

template<class T>

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

};

template<class T>

Matrix<T>::Matrix()

{

n = 1;

m = 1;

arr = new T \* [1];

arr[0] = new T[1];

arr[0][0] = 0;

}

template<class T>

Matrix<T>::Matrix(int size1, int size2, T\*\* arr\_)

{

n = size1;

m = size2;

arr = new T \* [n];

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

arr[i] = new T[m];

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

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

arr[i][j] = arr\_[i][j];

}

template<class T>

Matrix<T>::Matrix(int size1, int size2)

{

n = size1;

m = size2;

arr = new T \* [n];

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

arr[i] = new T[m];

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

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

arr[i][j] = rand() % 100;

}

template<class T>

Matrix<T>::Matrix(const Matrix<T>& other)

{

n = other.n;

m = other.m;

arr = new T \* [n];

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

arr[i] = new T[m];

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

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

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

}

template<class T>

Matrix<T>::~Matrix()

{

if (arr != 0)

{

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

delete[] arr[i];

delete[] arr;

arr = 0;

n = 0;

m = 0;

}

}

template<class T>

int Matrix<T>::GetN()

{

return n;

}

template<class T>

int Matrix<T>::GetM()

{

return m;

}

template<class T>

T\*\* Matrix<T>::GetArr()

{

return arr;

}

template<class T>

void Matrix<T>::Set(int \_n, int \_m, T\*\* \_arr)

{

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

delete[] arr[i];

delete[] arr;

n = \_n;

m = \_m;

arr = new T \* [n];

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

arr[i] = new T[m];

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

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

arr[i][j] = \_arr[i][j];

}

template<class T>

void Matrix<T>::SetI(int index1, int index2, T t)

{

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

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

if (i == index1 && j == index2)

arr[i][j] = t;

}

template<class T>

int Matrix<T>::Search1(int search)

{

int count = 0;

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

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

if (arr[i][j] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

float Matrix<T>::Search1(float search)

{

int count = 0;

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

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

if (arr[i][j] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

double Matrix<T>::Search1(double search)

{

int count = 0;

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

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

if (arr[i][j] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

void Matrix<T>::Search2(int search)

{

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

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

if (arr[i][j] == search)

cout << "Reference at " << i << j << " position" << endl;

}

template<class T>

void Matrix<T>::Search2(float search)

{

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

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

if (arr[i][j] == search)

cout << "Reference at " << i << j << " position" << endl;

}

template<class T>

void Matrix<T>::Search2(double search)

{

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

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

if (arr[i][j] == search)

cout << "Reference at " << i << j << " position" << endl;

}

template<class T>

Matrix<T> Matrix<T>::operator = (const Matrix<T>& other)

{

if (\*this == other)

return \*this;

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

delete[] arr[i];

delete[] arr;

this->n = other.n;

this->m = other.m;

arr = new T \* [n];

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

arr[i] = new T[m];

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

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

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

return \*this;

}

template<class T>

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

{

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

{

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

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

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

return false;

return true;

}

else

return false;

}

template<class T>

T\*& Matrix<T>::operator [](const int \_n)

{

return arr[\_n];

}

template<class T>

Matrix<T> Matrix<T>::operator +(const Matrix<T>& other)

{

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

{

Matrix<T> matrix(n, m, other.arr);

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

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

matrix.arr[i][j] = other.arr[i][j] + arr[i][j];

return matrix;

}

else

throw - 1;

}

template<class T>

Matrix<T> Matrix<T>::operator -(const Matrix<T>& other)

{

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

{

Matrix<T> matrix(n, m, other.arr);

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

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

matrix.arr[i][j] = arr[i][j] - other.arr[i][j];

return matrix;

}

else

throw - 1;

}

template<class T>

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

{

if (this->m == other.n)

{

Matrix<T> result;

result.n = this->n;

result.m = other.m;

result.arr = new T \* [result.n];

for (int i = 0; i < result.n; i++)

result.arr[i] = new T[result.m];

for (int i = 0; i < result.n; i++)

for (int j = 0; j < result.m; j++)

{

result.arr[i][j] = 0;

for (int k = 0; k < this->m; k++)

{

result.arr[i][j] = result.arr[i][j] + (arr[i][k] \* other.arr[k][j]);

}

}

return result;

}

else

throw - 1;

}

template<class T>

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

{

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

{

if (i > 0) os << endl;

os << "|";

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

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

os << "|";

}

return os;

}

template<class T>

istream& operator>>(istream& in, Matrix<T>& mat)

{

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

delete[] mat.arr[i];

delete[] mat.arr;

cout << "Vvedite razmer" << endl << "n = ";

in >> mat.n;

cout << "m = ";

in >> mat.m;

T\*\* temp;

temp = new T \* [mat.n];

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

temp[i] = new T[mat.m];

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

{

cout << "Vvedite " << i + 1 << " stroku" << endl;

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

in >> temp[i][j];

}

mat.arr = new T \* [mat.n];

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

mat.arr[i] = new T[mat.m];

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

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

mat.SetI(i, j, temp[i][j]);

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

delete[] temp[i];

delete[] temp;

return in;

}

**Vector.h**

#pragma once

#include <time.h>

#include "Matrix.h"

#include <iostream>

using namespace std;

template <class T>

class Vector

{

protected:

int N;

T\* arr;

public:

Vector();

Vector(int size, const T\* \_arr);

Vector(int size);

Vector(const Vector<T>& other);

~Vector();

int GetN();

T\* GetArr();

int Search1(int search);

float Search1(float search);

double Search1(double search);

void Search2(int search);

void Search2(float search);

void Search2(double search);

void Set(int n, T\* \_arr);

void SetI(int index, T t);

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

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

T operator [](const int i);

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

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

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

template <class T>

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

template <class T>

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

void Bubble\_sort();

void Insertion\_sort();

void Quick\_sort(int first, int last);

};

template<class T>

Vector<T>::Vector()

{

N = 1;

arr = new T[1];

arr[0] = 0;

}

template<class T>

Vector<T>::Vector(int size, const T\* arr\_)

{

N = size;

arr = new T[N];

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

arr[i] = arr\_[i];

}

Vector<int>::Vector(int size)

{

N = size;

arr = new int[N];

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

arr[i] = rand() % 100;

}

template<class T>

Vector<T>::Vector(const Vector<T>& other)

{

N = other.N;

arr = new T[N];

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

arr[i] = other.arr[i];

}

template<class T>

Vector<T>::~Vector()

{

if (arr != 0)

{

delete[] arr;

arr = 0;

N = 0;

}

}

template<class T>

int Vector<T>::GetN()

{

return N;

}

template<class T>

T\* Vector<T>::GetArr()

{

return arr;

}

template<class T>

int Vector<T>::Search1(int search)

{

int count = 0;

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

if (arr[i] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

float Vector<T>::Search1(float search)

{

int count = 0;

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

if (arr[i] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

double Vector<T>::Search1(double search)

{

int count = 0;

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

if (arr[i] == search)

count++;

cout << "Number of references is " << count << endl;

return 0;

}

template<class T>

void Vector<T>::Search2(int search)

{

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

if (arr[i] == search)

cout << "Reference at " << i << " position" << endl;

}

template<class T>

void Vector<T>::Search2(float search)

{

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

if (arr[i] == search)

cout << "Reference at " << i << " position" << endl;

}

template<class T>

void Vector<T>::Search2(double search)

{

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

if (arr[i] == search)

cout << "Reference at " << i << " position" << endl;

}

template<class T>

void Vector<T>::Set(int n, T\* \_arr)

{

delete[] arr;

N = n;

arr = new T[n];

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

arr[i] = \_arr[i];

}

template<class T>

void Vector<T>::SetI(int index, T t)

{

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

if (i == index)

arr[i] = t;

}

template<class T>

Vector<T> Vector<T>::operator=(const Vector<T>& other)

{

if (\*this == other)

return \*this;

delete[] arr;

this->N = other.N;

arr = new T[N];

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

arr[i] = other.arr[i];

return \*this;

}

template<class T>

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

{

if (N == other.N)

{

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

if (arr[i] != other.arr[i])

return false;

return true;

}

else

return false;

}

template<class T>

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

{

return arr[i];

}

template<class T>

Vector<T> Vector<T>::operator+(const Vector<T>& other)

{

if (N == other.N)

{

Vector<T> vector(N, other.arr);

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

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

return vector;

}

else

throw - 1;

}

template<class T>

Vector<T> Vector<T>::operator-(const Vector<T>& other)

{

if (N == other.N)

{

Vector<T> vector(N, other.arr);

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

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

return vector;

}

else

throw - 1;

}

template<class T>

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

{

T result = 0;

if (N == other.N)

{

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

result = result + (arr[i] \* other.arr[i]);

return result;

}

else

throw - 1;

}

template<class T>

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

{

os << "(";

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

{

os << vec.arr[i];

if (i < vec.N - 1)

os << " ";

}

os << ")^T";

return os;

}

template<class T>

istream& operator>>(istream& in, Vector<T>& vec)

{

cout << "Vvedite razmer" << endl << "N = ";

in >> vec.N;

T\* temp;

temp = new T[vec.N];

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

in >> temp[i];

Vector<T> V(vec.N, temp);

vec = V;

delete[] temp;

return in;

}

template<class T>

void Vector<T>::Bubble\_sort()

{

T temp = 0;

for (int i = 0; i < N - 1; i++)

for (int j = 0; j < N - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

temp = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = temp;

}

}

}

template<class T>

void Vector<T>::Insertion\_sort()

{

T temp;

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

{

int j = i;

while ((j > 0) && (arr[j - 1] < arr[j]))

{

temp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = temp;

j--;

}

}

}

template<class T>

void Vector<T>::Quick\_sort(int first, int last)

{

T mid, count;

int f = first, l = last;

mid = arr[(f + l) / 2];

do

{

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l)

{

count = arr[f];

arr[f] = arr[l];

arr[l] = count;

f++;

l--;

}

} while (f < l);

if (first < l) Quick\_sort(first, l);

if (f < last) Quick\_sort(f, last);

}

**Operations.h**

#pragma once

#include "Vector.h"

#include "Matrix.h"

template <class T>

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

{

if (matrix.GetN() == vector.GetN())

{

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

Vector<T> result(matrix.GetM(), arr);

for (int i = 0; i < result.GetN(); i++)

{

result.SetI(i, 0);

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

{

result.SetI(i, result[i] + (matrix[i][j] \* vector[j]));

}

}

delete[] arr;

return result;

}

else

throw - 1;

};

template <class T>

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

{

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

{

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

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

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

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

for (int i = 0; i < result.GetN(); i++)

for (int j = 0; j < result.GetM(); j++)

{

result.SetI(i, j, matrix[0][j] \* vector[i]);

}

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

delete[] arr[i];

delete[] arr;

return result;

}

else

throw - 1;

};

**main.cpp**

#include "Vector.h"

#include "Matrix.h"

#include "Operations.h"

#include <iostream>

#include <time.h>

#include <locale.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

int type;

int searchI;

float searchF;

double searchD;

cout << "1. Целочисленный" << endl << "2. Дробный" << endl << "3. Вещественный" << endl << "Пожалуйста, выберите тип данных для векторов: ";

cin >> type;

switch (type)

{

case (1):

{

Vector<int> V1;

Vector<int> V2;

Vector<int> V;

Vector<int> SV;

Matrix<int> M1;

Matrix<int> M2;

Matrix<int> M;

Matrix<int> SM;

int Scalar;

int type2;

cout << "1. вектор + вектор" << endl << "2. вектор - вектор" << endl << "3. вектор \* вектор" <<

endl << "4. матрица + матрица" << endl << "5. матрица - матрица" << endl << "6. матрица \* матрица" <<

endl << "7. вектор \* матрица" << endl << "8. матрица \* вектор" << endl << "9. Узнать количество и места всех вхождений заданного числа в матрице" << endl << "10. Узнать количество и места всех вхождений заданного числа в векторе" << endl << "Выберите операцию: ";

cin >> type2;

switch (type2)

{

case (1):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 + V2;

cout << V1 << " + " << V2 << " = " << V << endl;

break;

case (2):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 - V2;

cout << V1 << " - " << V2 << " = " << V << endl;

break;

case (3):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

Scalar = V1 \* V2;

cout << V1 << " \* " << V2 << " = " << Scalar << endl;

break;

case (4):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 + M2;

cout << M1 << endl << " + " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (5):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 - M2;

cout << M1 << endl << " - " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (6):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 \* M2;

cout << M1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (7):

cout << "Введите вектор" << endl;

cin >> V1;

cout << "Введите матрицу" << endl;

cin >> M2;

M = V1 \* M2;

cout << V1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (8):

cout << "Введите матрицу" << endl;

cin >> M1;

cout << "Введите вектор" << endl;

cin >> V2;

V = M1 \* V2;

cout << M1 << endl << " \* " << endl << V2 << endl << " = " << endl << V << endl;

break;

case (9):

cout << "Введите матрицу" << endl;

cin >> SM;

cout << "Введите число" << endl;

cin >> searchI;

SM.Search1(searchI);

SM.Search2(searchI);

break;

case (10):

cout << "введите вектор" << endl;

cin >> SV;

cout << "Введите число" << endl;

cin >> searchI;

SV.Search1(searchI);

SV.Search2(searchI);

break;

}

}

break;

case(2):

{

Vector<float> V1;

Vector<float> V2;

Vector<float> V;

Vector<float> SV;

Matrix<float> M1;

Matrix<float> M2;

Matrix<float> M;

Matrix<float> SM;

float Scalar;

int type2;

cout << "1. вектор + вектор" << endl << "2. вектор - вектор" << endl << "3. вектор \* вектор" <<

endl << "4. матрица + матрица" << endl << "5. матрица - матрица" << endl << "6. матрица \* матрица" <<

endl << "7. вектор \* матрица" << endl << "8. матрица \* вектор" << endl << "9. Узнать количество и места всех вхождений заданного числа в матрице" << endl << "10. Узнать количество и места всех вхождений заданного числа в векторе" << endl << "Выберите операцию: ";

cin >> type2;

switch (type2)

{

case (1):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 + V2;

cout << V1 << " + " << V2 << " = " << V << endl;

break;

case (2):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 - V2;

cout << V1 << " - " << V2 << " = " << V << endl;

break;

case (3):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

Scalar = V1 \* V2;

cout << V1 << " \* " << V2 << " = " << Scalar << endl;

break;

case (4):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 + M2;

cout << M1 << endl << " + " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (5):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 - M2;

cout << M1 << endl << " - " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (6):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 \* M2;

cout << M1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (7):

cout << "Введите вектор" << endl;

cin >> V1;

cout << "Введите матрицу" << endl;

cin >> M2;

M = V1 \* M2;

cout << V1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (8):

cout << "Введите матрицу" << endl;

cin >> M1;

cout << "Введите вектор" << endl;

cin >> V2;

V = M1 \* V2;

cout << M1 << endl << " \* " << endl << V2 << endl << " = " << endl << V << endl;

break;

case (9):

cout << "Введите матрицу" << endl;

cin >> SM;

cout << "Введите число" << endl;

cin >> searchF;

SM.Search1(searchF);

SM.Search2(searchF);

break;

case (10):

cout << "введите вектор" << endl;

cin >> SV;

cout << "Введите число" << endl;

cin >> searchF;

SV.Search1(searchF);

SV.Search2(searchF);

break;

}

}

break;

case(3):

{

Vector<double> V1;

Vector<double> V2;

Vector<double> V;

Vector<double> SV;

Matrix<double> M1;

Matrix<double> M2;

Matrix<double> M;

Matrix<double> SM;

double Scalar;

int type2;

cout << "1. вектор + вектор" << endl << "2. вектор - вектор" << endl << "3. вектор \* вектор" <<

endl << "4. матрица + матрица" << endl << "5. матрица - матрица" << endl << "6. матрица \* матрица" <<

endl << "7. вектор \* матрица" << endl << "8. матрица \* вектор" << endl << "9. Узнать количество и места всех вхождений заданного числа в матрице" << endl << "10. Узнать количество и места всех вхождений заданного числа в векторе" << endl << "Выберите операцию: ";

cin >> type2;

switch (type2)

{

case (1):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 + V2;

cout << V1 << " + " << V2 << " = " << V << endl;

break;

case (2):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

V = V1 - V2;

cout << V1 << " - " << V2 << " = " << V << endl;

break;

case (3):

cout << "Введите первый вектор" << endl;

cin >> V1;

cout << "Введите второй вектор" << endl;

cin >> V2;

Scalar = V1 \* V2;

cout << V1 << " \* " << V2 << " = " << Scalar << endl;

break;

case (4):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 + M2;

cout << M1 << endl << " + " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (5):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 - M2;

cout << M1 << endl << " - " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (6):

cout << "Введите первую матрицу" << endl;

cin >> M1;

cout << "Введите вторую матрицу" << endl;

cin >> M2;

M = M1 \* M2;

cout << M1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (7):

cout << "введите вектор" << endl;

cin >> V1;

cout << "Введите матрицу" << endl;

cin >> M2;

M = V1 \* M2;

cout << V1 << endl << " \* " << endl << M2 << endl << " = " << endl << M << endl;

break;

case (8):

cout << "Введите матрицу" << endl;

cin >> M1;

cout << "Введите вектор" << endl;

cin >> V2;

V = M1 \* V2;

cout << M1 << endl << " \* " << endl << V2 << endl << " = " << endl << V << endl;

break;

case (9):

cout << "Введите матрицу" << endl;

cin >> SM;

cout << "Введите число" << endl;

cin >> searchD;

SM.Search1(searchD);

SM.Search2(searchD);

break;

case (10):

cout << "введите вектор" << endl;

cin >> SV;

cout << "Введите число" << endl;

cin >> searchD;

SV.Search1(searchD);

SV.Search2(searchD);

break;

}

}

break;

default:

throw - 1;

}

/\*

double t = clock();

double seconds = clock() - t;

cout << seconds;

\*/

return 0;

}