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

Мурзинов Антон Денисович

Проверил:

ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc24628114)

[Постановка задачи 4](#_Toc24628115)

[Руководство пользователя 5](#_Toc24628116)

[Руководство программиста 6](#_Toc24628117)-10

[Эксперименты 11-1](#_Toc24628118)3

[Заключение 1](#_Toc24628119)4

[Литература 1](#_Toc24628120)5

[Приложение 16-2](#_Toc24628121)9

# 1.Введение

Число является одним из основных понятий математики. Понятие числа развивалось в тесной связи с изучением величин; эта связь сохраняется и теперь. Во всех разделах современной математики и информатики приходится рассматривать разные величины и пользоваться числами.

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. 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 размерn {\displaystyle n}
* умножение векторов, имеющих один и тот же размер

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 n {\displaystyle n} столбцов, можно умножить справа на матрицу, имеющую n {\displaystyle n} n строк);
* n {\displaystyle n} n {\displaystyle n} 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. Запустить консоль программы «MatrixAndVectors.exe».
2. Создать объект типа vector или matrix.
3. Произвести все необходимые операции над вектором или матрицей.

# 

# 4.Руководство программиста

***Описание структуры программы***

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

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

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

В «matrix.h» определен класс Matrix.

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

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

***Описание структуры программы***

В программе определены шаблонные классы VECTOR и Matric.

Внутри класса Matric определены следующие поля:

• M\*\* Matrix – шаблонный двойной указатель

• int m – количество строк

• int n – количество столбцов

Внутри класса matrix определены следующие поля:

• 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[A]& 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

• A\*\* getMatrix() – метод, возвращающий массив

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

• void setMatrix(A\*\* e) – метод, устанавливающий значение массива в матрицу

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

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

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

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

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

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

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

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

Private поле:

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

Внутри класса VECTOR определены следующие поля:

• Vector\* array – шаблонный указатель(массив)

• int size – размерность вектора

Внутри класса VECTOR определены следующие поля:

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

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

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

• ~ VECTOR () – деструктор

• int getSize() – метод, возвращающий размерность вектора

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

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

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

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

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

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

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

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

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

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

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

• VECTOR operator /(VECTOR & 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**

Вначале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. На каждом шаге сортировки сравнивается текущий элемент со всеми элементами в отсортированной части. Алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива.

1. **Быстрая сортировка**

Выбирается опорный элемент, который разбивает массив примерно на две равные части. Дальше массив сортируется таким образом, что все элементы, которые меньше опорного элемента, встают слева, а которые больше, справа. Дальше выбирается снова опорный элемент для подмассива, который оказался слева от первоначального опорного элемента и то же самое для правого. С помощью рекурсивной сортировки каждой части массива, программа выдает полностью отсортированный массив.

# 

# 5.Эксперименты

# Эксперименты

Оценим время, которое занимают матричные и векторно-матричные операции, рассмотрим код, отвечающий за суммирование матриц:

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

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

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

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

return F;

Асимптотическая сложность сложения:

Замеры сложения матриц:

|  |  |
| --- | --- |
| **Количество элементов в матрице** | **Время выполнения (сек.)** |
| **1000** | **0.60** |
| **2000** | **2.34** |
| **4000** | **9.1** |

Таблица 1: Время суммирования матриц.

Заметим, что 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 < mas.m; i++)

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

{

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

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

{

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

}

}

Асимптотическая сложность умножения:

Замеры умножения матриц:

|  |  |
| --- | --- |
| **Количество элементов в матрице** | **Время выполнения (сек.)** |
| **1000** | **9.42** |
| **2000** | **77.31** |
| **4000** | **6056.3** |

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. Оценим асимптотическую сложность векторно-матричного умножения:

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

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

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

{

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

}

Асимптотическая сложность :

Замеры умножения матриц и векторов:

|  |  |
| --- | --- |
| **Количество элементов в матрице** | **Время выполнения (сек.)** |
| **1000** | **0.002** |
| **2000** | **0.013** |
| **4000** | **0.055** |

Таблица 3: Время умножения матриц и векторов

Заметим, что при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

Для стандартных алгоритмов сортировки (BubbleSort, IsertionSort и QuickSort) асимптотическая сложность уже определена. Проведем замеры:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Bubble Sort** | **Insertion Sort** | **Quick Sort** |
| **100 000 элементов** | **13.97 сек** | **1.22 сек** | **0.011 сек** |
| **200 000 элементов** | **60.13 сек** | **7.32 сек** | **0.023 сек** |
| **400 000 элементов** | **246.32 сек** | **30.22 сек** | **0.055 сек** |
| **Асимптотическая сложность** |  |  |  |

Таблица 4: Время работы сортировок

Можно заметить, что затрачиваемое на реализацию операции время, растет эквивалентно асимптотической функции.

# 6.Заключение

В ходе лабораторной работы была написана программа, позволяющая работать с векторами и матрицами, а именно выполнять все алгебраические операции с использованием шаблонов.

Эксперименты показали, что то время которое затрачивается в теории на реализацию матричных, матрично-векторных операций с некоторыми погрешностями совпадает с асимптотической сложности времени, которого мы ожидали.

# 

# 7.Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003
2. <https://ru.wikipedia.org>
3. Официальный сайт Habr. – Режим доступа <https://habr.com/ru/post/339656/>
4. Официальный сайт Microsoft. – Режим доступа <https://docs.microsoft.com/ru-ru/cpp/c-language/main-function-and-program-execution?view=vs-2019>

# 

# 8.Приложение

***Приложение 1. Код программы***

**Main.cpp**

#include <malloc.h>

#include <vector>

#include <iostream>

#include <iomanip>

#include"main.h"

#include"Matrix.h"

#include"Vector.h"

using namespace std;

int main()

{

cout << "Create Vector 1" << endl;

VECTORr<int> A;

cin >> A;

cout << "Return Vector 1" << endl;

cout << A << "\n";

int K = 0;

int arr[] = { 8,2,1,4,8,6,7 };

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

VECTOR<int> B(7, arr); //создадим два вектора одинаковой размерности для операций над ними

VECTOR<int> C(7, arrr);

VECTOR<int> D;

cout << "Addition" << endl;

D = B + C;

cout << D << "\n";

cout << "Multiplication" << endl;

K = B \* C;

cout << K << "\n";

cout << "Subtraction" << endl;

D = B - C;

cout << D << "\n";

cout << "Division" << endl;

D = C / B;

cout << D << "\n";

cout << "Sorts" << endl << endl;

cout << "BubbleSort" << endl;

int t = clock();

int arrr1[] = { 9,2,3,4,5,6,7 };

VECTOR<int> AH(7, arrr1);

t = AH.BubbleSort();

cout << "Time sort" << t << endl;

cout << "Sorted vector" << endl;

cout << AH << endl << endl;

int arrr2[] = { 9,2,1,4,5,6,3 };

VECTOR<int> U(7, arrr2);

cout << "QuickSort" << endl;

int o = clock();

o = U.QuickSort(0, B.GetRazmer() - 1);

cout << "Time sort" << o << endl;

cout << "Sorted vector" << endl;

cout << U << "\n" << "\n";

int arrr3[] = { 9,2,3,2,5,6,4 };

VECTOR<int> E(7, arrr3);

cout << "InsertionSort" << endl;

int w = clock();

w = E.InsertionSort();

cout << "Time sort" << w << endl;

cout << "Sorted vector" << end;;

cout << E << endl << endl;

cout << "Matrix 1 " << endl;

Matrix<int> AB;

cin >> AB;

cout << "Square Matrix 1" << AB << endl;

cout << "Enter dimension";

cout << "Matrica 2" << endl;

Matrix<int> AC;

cin >> AC;

cout << "Matrix 2" << AB << endl;

Matrix<int> ABC;

cout << "Addition" << endl;

ABC = AB + AC;

cout << ABC << endl;

cout << "Multiplication" << endl;

ABC = AB \* AC;

cout << ABC << endl;

cout << "Division" << endl;

ABC = AB - AC;

cout << ABC << endl;

cout << "The number of occurrences of the specified value" << endl;

cout << "Multiplication Vector\*Matrix" << endl;

int arrar[] = { 1,2,3 };

VECTOR<int> V1(3, arrar);

Matrix<int> V2;

Matrix<int> V3;

cout << "Enter dimention 1\*3" << endl;

cin >> V2;

V3 = V1 \* V2;

cout << V3 << endl;

cout << "Multiplication Matrix\*Vector" << endl;

Matrix<int> VV;

cout << "Enter dimention 3\*3" << endl;

cin >> VV;

int arrr4[] = { 9,2,1 };

VECTOR<int> L(3, arrr4);

VECTOR<int> I;

I = VV \* L;

cout << I << endl;

int K;

K = E.FindVector(3);

cout << K << endl;

int F;

F = AB.FindVector(2);

cout << F << endl;

return 0;

**main.h:**

#include "Vector.h"

#include "Matrix.h"

template <class T>

VECTOR<T> operator \*(Matrix<T>& matrix, VECTOR<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]);

}

}

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

delete[] arr;

return result;

}

else

throw - 1;

};

template <class T>

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

{

if (matrix.GetN() == vector.Size() && 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;

}

else

throw - 1;

};

**Vector.h:**

#include <iostream>

#include <clocale>

#include <iomanip>

#include"Matrix.h"

using namespace std;

template <class VECTOR>

class VECTOR

{

protected:

int size;

Vector\* array;

public:

VECTOR();

VECTOR(int r, Vector\* A);

VECTOR(VECTOR& B);

~VECTOR();

Vector\* GetVector();

int GetSize();

void SetVector(Vector\* C);

void SetSize(int e);

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

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

Vector operator [](const int i);

VECTOR<Vector> operator + (const VECTORr& other);

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

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

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

template <class Vector>

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

template <class Vector>

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

clock\_t BubbleSort();

clock\_t InsertionSort();

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

int FindVector(Vector a); //количество вхождений указанного значения

};

template<class Vector>

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

{

array = nullptr;

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 << "Imput error array = 0 " << endl;

exit(0);

}

}

template<class Vector>

VECTOR<Vector>::VECTOR(VECTOR& B)

{

array = nullptr;

size = B.GetSize();

array = new Vector[size];

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

{

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

}

}

template<class Vector>

Vector\* VECTOR<Vector>::GetVector()

{

return array;

}

template<class Vector>

int VECTOR<Vector>::GetSize()

{

return size;

}

template<class Vector>

void VECTOR<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<class Vector>

VECTOR<Vector>::VECTOR()

{

size = 1;

array = new Vector[1];

Vector a = 0;

array[0] = a;

}

template<class Vector>

VECTOR<Vector>::~VECTOR()

{

if (array != 0)

{

array = 0;

delete[] array;

}

size = 0;

}

template<class Vector>

VECTOR<Vector> VECTOR<Vector>::operator=(const VECTOR<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<class Vector>

bool VECTOR<Vector>::operator==(const VECTOR<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<class Vector>

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

{

return array[i];

}

template<class Vector>

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

{

if (size == other.size)

{

VECTOR<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<class Vector>

void VECTOR<Vector>::SetSize(int e)

{

size = e;

}

template<class Vector>

VECTOR<Vector> VECTOR<Vector>::operator-(const VECTOR<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<class Vector>

Vector VECTOR<Vector>::operator \*(const VECTOR<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;

}

else

throw - 1;

}

template<class Vector>

VECTOR<Vector> VECTOR<Vector>::operator /(VECTOR<Vector>& other)

{

if (size != other.GetSize())throw length\_error("Imput error");

VECTOR<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<class Vector>

ostream& operator<<(ostream& os, const VECTOR<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<class Vector>

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

{

cout << "Enter size" << endl;

in >> vec.size;

Vector\* mas;

mas = new Vector[vec.size];

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

{

cout << "[" << i << "]" << endl;

in >> mas[i];

}

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

vec = V;

mas = 0;

delete[] mas;

return in;

}

template<class Vector>

clock\_t VECTOR<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<class Vector>

int VECTOR<Vector>::FindVector(Vector a) //количество вхождений указанного значения

{

int amount = 0;

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

{

if (array[i] == a)

{

amount = amount + 1;

}

}

return amount;

}

template<class Vector>

clock\_t VECTOR<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<class Vector>

clock\_t VECTOR<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<class Vector>

inline int VECTOR<Vector>::FindVector(Vector a)

{

return 0;

}

**Matrix.h**

#include<iostream>

#include <clocale>

#include <iomanip>

#include"Vector.h"

using namespace std;

template<class M>

class Matrix

{

private:

void MemorySize(int width, int length) //выделение памяти

{

array = new M \* [m];

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

}

protected:

int m; //строки

int n; //столбцы

M\*\* array;

public:

Matrix();

Matrix(int a, int b); //пустая матрица нужной размерности

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

Matrix(const Matrix<M>& other);

~Matrix();

M\*\* GetMatrix(); //массив матрицы

int GetM(); //количество строк

int GetN(); //количество столбцов

void SetDimension(int width, int length); //установить размерность

void SetMatrix(M\*\* e);

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

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

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

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

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

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

template<class M>

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

template<class M>

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

int FindVector(M a); //количество вхождений указанного значения

};

template<class M>

Matrix<M>::Matrix()

{

m = 1;

n = 1;

MemorySize(m, n);

array[0][0] = 0;

}

template<class M>

Matric<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<class 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<class 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<class 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<class M>

int Matrix<M>::FindVector(M a) //количество вхождений указанного значения

{

int amount = 0;

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

{

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

{

if (massiv[i][j] == a)

{

amount = amount + 1;

}

}

}

return kol;

}

template<class M>

int Matric<M>::GetM()

{

return m;

}

template<class M>

int Matrix<M>::GetN()

{

return n;

}

template<class M>

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

{

return array;

}

template<class M>

void Matrix<M>::SetDimention(int width, int length) //установить размерность

{

m = width;

n = length;

}

template<class M>

void Matrix<M>::SetMatrix(M\*\* e) //переписать значение полей в матрице уже созданной размерности

{

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

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

array[i][j] = e[i][j]; //переписываем массив новых значений

}

template<class M>

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

{

if (\*this == other)

return \*this;

for (int i = 0; i < m; i++) //удаляем старый массив, чтобы можно было присваивать матрицу другой размерности

{

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<class M>

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

{

return array[i];

}

template<class 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<class 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;

}

else

throw - 1;

}

template<class 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<class M>

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

{

if (n == other.m)

{

Matrix<M> mas(m, (other.n)); //Делаю пустой массив нужной размерности

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

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

{

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

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

{

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

}

}

return mas;

}

else

throw - 1;

}

template<class M>

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

{

cout << endl;

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

{

if (i > 0) os << endl;

os << "|";

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

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

os << "|";

}

return os;

}

template<class M>

istream& operator>>(istream& in, Matrix<M>& mas)

{

if ((mas.GetM() != 0) || (mas.GetN() != 0))

{

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

delete[] mat.array[i];

delete[] mat.array;

}

cout << "Enter size" << endl;

cout << "m = " << endl;

cout << "n = " << endl;

in >> mas.n;

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

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

{

cout << "Enter " << i + 1 << " line" << endl;

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

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

}

return in;

}