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**

Выполнила:

студент ф-та ИТММ ПМИ – 381903-3

Микийчук Дмитрий Алексеевич

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

**Матрица** — [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82), записываемый в виде прямоугольной таблицы элементов [кольца](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) или [поля](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) (например, [целых](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) действительных или [комплексных](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы[[1]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)#cite_note-1), в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Матрицы широко применяются в математике для компактной записи систем линейных [алгебраических](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5) или [дифференциальных](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение [систем линейных уравнений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9) сводится к операциям над матрицами.

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) матриц, 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 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)).

**Вектор**  — в простейшем случае [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82), характеризующийся величиной и направлением. Например, в геометрии и в естественных науках вектор есть направленный отрезок прямой в [евклидовом пространстве](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%BE%D0%B2%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) (или на плоскости)

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

# 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 программу.
2. Создать объект типа vector или matrix.
3. Произвести операции над вектором и матрицей.

# 4.Руководство программиста

***Описание структуры программы***

Программа состоит из одного решения, которое называется «lab\_vector».

В решении содержится 4 элементов: «Vector.h», «Header.h», «Matrix.h», «main.cpp».

В «Vector.h» определен класс Vector**.**

В «matrix.h» определен класс Matrix.

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 Vector и Matrix.

Внутри класса Matrix определены следующие поля(protected):

• M\*\* Matrix – шаблонный двойной указатель

• int m – количество строк

• int n – количество столбцов

Внутри класса matrix определены следующие поля(public):

• Matrix() – конструктор по умолчанию

• Matrix(int A, int B) – конструктор инциализатор, который устанавливает параметры размеров

• Matrix(int C, int D, M\*\* arr) – конструктор инциализатор, устанавливает размеры и массив

• Matrix(const Matrix<M>& Znach) – конструктор копирования, принимает на вход объект, создает объект с теми же характеристиками, что и переданный, и той же матрицей

• ~Matrix() – деструктор

• int Getm() – метод, возвращающий количество строк

• int Getn() – метод, возвращающий количество столбцов

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

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

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

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

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

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

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

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

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

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

• void MSize(int, int) - выделяет память до размеров матрицы;

Внутри класса Vector определены следующие поля(protected):

• int size – размерность вектора

Внутри класса Vector определены следующие поля(public):

• Vector() – конструктор по умолчанию

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

• Vector(const Vector<V>& Znach) – конструктор копирования, принимает на вход объект типа Vector, создает объект с теми же характеристиками, что и переданный

• ~Vector() – деструктор

• int getSize() – метод, возвращающий размерность вектора

• V\* getVector() – метод, возвращающий вектор

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

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

• Vector operator +(const Vector & Znach) - перегрузка оператора суммы

• Vector operator –(const Vector & Znach) - перегрузка оператора ­разности

• Vector operator \*(const Vector & Znach) - перегрузка оператора умножения

• Vector operator =(const Vector & Znach) - перегрузка оператора равенства

• bool operator ==(const Vector & Znach) - 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

• Vector operator /(Vector& Znach) – 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**

Принцип действий прост: обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть массива, запихивая найденные максимумы в конец.

1. **Сортировка вставкой**

Запомнить во временную переменную значение текущего элемента массива;

Пока элементы слева от запомненного значения больше чем запомненное – перемещаем их на позицию вправо. Получается, что предыдущий элемент займет ячейку запомненного. А тот, что стоит перед предыдущим – переместится в свою очередь на место предыдущего. И так  элементы будут двигаться друг за дружкой.

Движение элементов заканчивается, если очередной элемент, который нужно сдвинуть, оказывается по значению меньше, чем тот, что запомнили во временную переменную в начале цикла.

Цикл берет следующий элемент, и опять сдвигает все, которые расположены перед ним и большие по значению.

1. **Быстрая сортировка**

Выбрать опорный элемент в массиве — часто встречается вариант с центральным элементом.

Разделить массив на две части следующим образом: все элементы из левой части, которые больше или равны опорному, перекидываем в правую, аналогично, все элементы из правой, которые меньше или равны опорному кидаем в левую часть.

В результате предыдущего шага в левой части массива останутся элементы, которые меньше или равны центральному, а в правой — больше либо равны.

Рекурсивно повторяем действие для левой и правой части массива.

Заходы в рекурсию прекратятся в тот момент, когда размер обоих частей будет меньше или равен единице.

5.Эксперимент

# Эксперименты

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

1. Рассмотрим код отвечающий за суммирование матриц:

Matrix<M> res (m, n, Znach.mass);//создание матрицы

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

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

{

res.mass[i][j] += Znach.mass[i][j]

}

Return res;

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

Таблица 1: Время суммирования матриц.

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

1. Теперь аналогично оценим асимптотическую сложность матричного умножения:

|  |
| --- |
| Matrix<M>proiz(m, Znach.n); |
|  | for (int i = 0; i < proiz.m; i++) |
|  | for (int j = 0; j < proiz.n; j++) |
|  | { |
|  | proiz.mass[i][j] = 0; |
|  | for (int k; k < n; k++) |
|  | { |
|  | proiz.mass[i][j] = proiz.mass[i][j] + (proiz.mass[i][k] \* proiz.mass[k][j]); |
|  | } |
|  | } |
|  | return proiz; |
|  | } |
|  | else |
|  | return throw ("incorrect row or column values"); |
|  | } |
| Количество элементов в матрице | | | Время выполнения (сек.) |
| 1000 | | | 9.40 |
| 2000 | | | 79.37 |
| 4000 | | | 6034.6 |

Таблица 2: Время умножения матриц.

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

|  |
| --- |
| 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++) |
|  | { |
|  | /\* result.SetRazmernost(i, j); |
|  | result.s[i]=(matrix[0][j] \* vector[i]);\*/ |
|  | arr[i][j] = (matrix[0][j] \* vector[i]); |
|  | } |
|  | Matrix<T> res(matrix.Getn(), vector.GetSize(), arr); |
|  |  |
|  |  |
|  | for (int i = 0; i < vector.GetSize(); i++) |
|  | delete[] arr[i]; |
|  | delete[] arr; |
|  | return res; |
| Количество элементов в матрице | | | Время выполнения (сек.) |
| 1000 | | | 0.002 |
| 2000 | | | 0.014 |
| 4000 | | | 0.053 |

Таблица 3: Время умножения матриц и векторов

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сортировка пузырьком | Сортировка вставками | Быстрая сортировка  (Хоара) |
| 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

# 6.Заключение

Я написал программу, которая выполняет 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.