МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Разработка структуры хранения матриц специального вида»**

**Выполнил:** студент группы 381903-3

Балашов Максим Алесеевич

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

Нижний Новгород

2020.

Содержание

[1. Введение 3](#_Toc560926)

[2. Постановка задачи 4](#_Toc560927)

[3. Руководство пользователя 5](#_Toc560928)

[4. Руководство программиста 6](#_Toc560929)

[4.1 Описание структуры программы 6](#_Toc560930)

[4.2 Описание структур данных 6](#_Toc560931)

[4.3 Описание алгоритмов 7](#_Toc560932)

[5. Заключение 10](#_Toc560933)

[6. Литература 11](#_Toc560934)

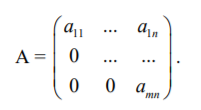
# Введение

Матрица – это таблица из m строк и n столбцов. Элементами матрицы могут быть объекты совершенно разнообразной природы: числа, переменные или, к примеру, иные матрицы.

Матрицы широко применяются в математике и физике для компактной записи и решения систем линейных алгебраических уравнений и систем дифференциальных уравнений. При этом количество строк матрицы соответствует количеству уравнений системы, а количество столбцов — количеству неизвестных величин. Матричный аппарат позволяет существенно упростить решение СЛАУ сведя его к операциям над матрицами.

Помимо матриц общего вида (прямоугольных) в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные и т.д.). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры. И с учетом этого они могут оказать более эффективными.

В данной лабораторной работе будут рассмотрены верхнетреугольные матрицы. Верхнетреугольной называется матрица, в которой все элементы под главной диагональю равны нулю:



# Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц специального вида (верхнетреугольные матрицы) и выполнение основных операций над ними:

* сложение/вычитание;
* умножение;
* копирование;
* сравнение.

Программные средства должны содержать:

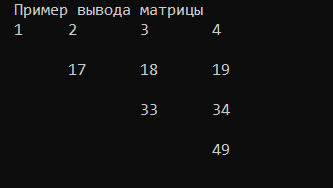
* класс Вектор (с использованием шаблона);
* класс Матрица (с использованием шаблона).

В классах 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), (вектор +-\* скаляр), матриц (матрица +-\* матрица).

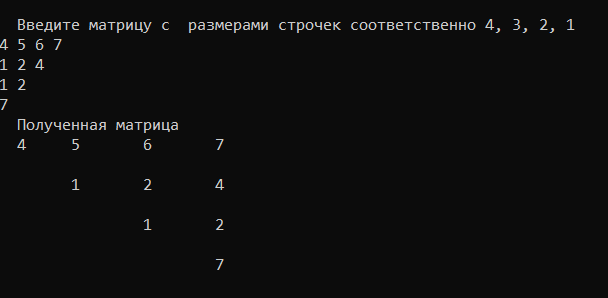
# Руководство пользователя

Программа начинает свою работу с вывода на консоль автоматически созданную матрицу.



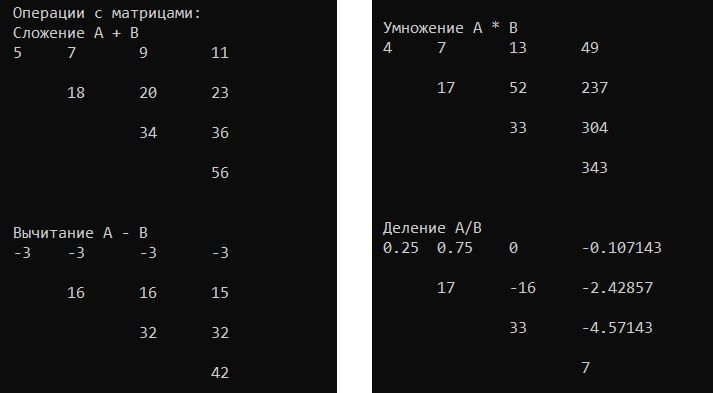
*Рисунок 1. Вывод на консоль матрицы.*

Следующим шагом программа запрашивает у пользователя ввести свою матрицу заданного размерами векторов. И выводит ее на консоль.



*Рисунок 2. Пример ввода матрицы пользователем.*

Последним шагом программа выполняет матричные операции с полученными данными, а именно: сложение, вычитание, умножение и деление матриц.

**

*Рисунок 3. Операции с матрицами.*

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* В модуле main.cpp определена стандартная функция int main(), в которой пользователю предлагается написать программу для осуществления операций с объектами типа TMatrix и TVector.
* В модуле TMatrix.h определен класс Matrix, в котором объявлены и определены его методы.
* В модуле TVector.h определен класс Vector, в котором объявлены и определены его методы.
* В модулях test\_main.cpp, test\_Matrix.cpp, test\_Vector.cpp написаны тесты на основе модулей main.cpp, Vector.h, Matrix.h соответственно

## Описание структур данных

#### Класс TVector.

Защищенные поля:

* ValType\* pVector;
* int Size; //размер вектора
* int StartIndex; // стартовый индекс

Содержит публичные методы:

* TVector(const TVector& v); //конструктор копирования
* TVector(int s = 10, int si = 0); //конструктор инициализации
* ~TVector();
* int GetSize(); // получить размер вектора
* int GetStartIndex(); // получить стартовый индекс
* ValType& val & operator[](int pos); //индексация
* bool operator==(const TVector& v); //сравнение
* bool operator!=(const TVector& v); //сравнение
* TVector operator\*(const ValType& val); // умножение на число
* TVector operator+(const ValType& val); //прибавление числа
* TVector operator-(const ValType& val); //вычитание числа
* TVector operator+(const TVector& v); //сложение
* TVector operator-(const TVector& v); //вычитание
* ValType operator\*(const TVector& v); //умножение

**Класс TMatrix.**

Полей у данного класса нет, так как он наследуется от класса TVector.

Содержит публичные методы:

* TMatrix(int s= 10);
* TMatrix(const TMatrix& mt);
* TMatrix(const TVector<TVector<ValType> >& mt); // конструктор преобразования типа
* bool operator==(const TMatrix& mt); //сравнение
* bool operator!=(const TMatrix& mt); //сравнение
* TMatrix& operator=(const TMatrix& mt); //присваивание
* TMatrix operator+(const TMatrix& mt); //сложение матриц
* TMatrix operator-(const TMatrix& mt); // вычитание

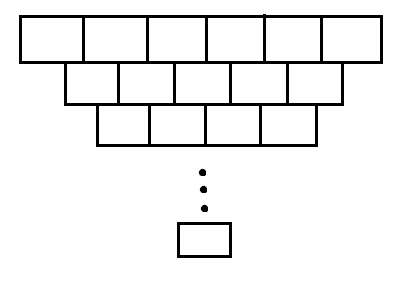
## Описание алгоритмов

Рассмотрим некоторые алгоритмы для класса TMatrix:

**Создание матрицы – конструктор инициализации, конструктор копирования и конструктор преобразования типа.**

* Конструктор инициализации.

Этот конструктор вызывает конструктор-иницализатор класса TVector с шаблоном <ValType> равным TVector<ValType>. Таким образом создается вектор, обозначим его за v, элементами которого также являются векторы, заданного размера, который принимает конструктор. После чего каждому элементу v, а именно векторам, присваивается размер, начиная от размера самой матрицы и уменьшая на единицу, и его стартовый индекс с помощью того же конструктора класса TVector, но с шаблоном <ValType>. В итоге получается треугольная матрица.

Первый вектор вектора v.

n-ый вектор вектора v.

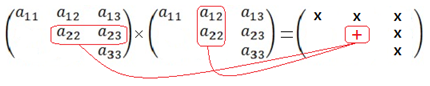
* Конструктор копирования.

Для того, чтобы реализовать этот конструктор, достаточно вызвать конструктор копирования класса TVector c шаблоном TVector<ValType>.

* Конструктор преобразования типа.

Работает аналогично, только этот конструктор принимает в качестве параметра не элемент класса TMatrix<ValType>, а элемент класса TVector<TVector<ValType>>.

**Умножение матриц.**

Произведением матриц A и B называется такая матрица С, элементы которой находятся по такому правилу: сij равен сумме произведений элементов i-той строки матрицы A на соответствующие элементы j-го столбца матрицы B . Таким образом, умножение осуществляется по правилу умножения строки на столбец.

*Рисунок 4. Схематическое нахождение произведения матриц.*

# Эксперименты

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

1. Рассмотрим код отвечающий за суммирование матриц:

Matrix operator+(Matrix &mp)

{

Matrix result(n,Size);

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

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

result.pVector[i][j]=pVector[i][j]+mp.pVector[i][j];

return result;

}

Асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.73 |
| 2000 | 2.24 |
| 4000 | 9.03 |

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:

Matrix operator\*(Matrix &mp)

{

Matrix result(n,mp.Size);

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

for(int j = 0; j<mp.Size; ++j)

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

result.pVector[i][j] += pVector[i][k]\*mp.pVector[k][j];

return result;

}

Асимптотическая сложность умножения:

Теперь мы произведем замеры времени умножения матриц

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 10.03 |
| 2000 | 80.66 |
| 4000 | 6045.4 |

Таблица 2: Время умножения матриц.

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

# Заключение

В ходе выполнения данной лабораторной была реализована структура хранения матриц специального вида, а именно верхнетреугольной. Были реализованы методы, позволяющие выполнять операции с векторами и матрицами, а именно: сложение, вычитание, умножение векторов/матриц и деление матриц.

Таким образом, мы рассмотрели виды матриц, операции над ними и их применение в разных областях.

Были разработаны тесты, проверяющие корректную работу методов данных классов, с помощью Google C++ Testing Framework.

# Литература

1. Применение матрицы [Электронный ресурс]

[http://topuch.ru/interesultnie-primeneniya-matrici-v-matematike-fizike-ekonomike-b/index.html](http://topuch.ru/interesnie-primeneniya-matrici-v-matematike-fizike-ekonomike-b/index.html)

1. Матрицы и их виды [Электронный ресурс]

<https://math1.ru/education/matrix/matrix.html>

1. Учебно-методическое пособие [Электронный ресурс]

<http://www.unn.ru/books/met_files/Pract_ADS.pdf>

1. Умножение матриц [Электронный ресурс]

<http://ru.solverbook.com/spravochnik/matricy/umnozhenie-matric/>