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**

**«Верхнетреугольная матрица»**

**Выполнил:** студент группы 381706-1

Корнев Никита Алексеевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533799739)

[2. Постановка задачи 4](#_Toc533799740)

[3. Руководство пользователя 5](#_Toc533799741)

[4. Руководство программиста 6](#_Toc533799742)

[4.1. Описание структуры программы 6](#_Toc533799743)

[4.2. Описание структур данных 7](#_Toc533799744)

[4.3 Описание алгоритмов 8](#_Toc533799745)

[5. Заключение 9](#_Toc533799746)

[6. Литература 10](#_Toc533799747)

# Введение

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

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. Хотя исторически рассматривались, например, треугольные матрицы, 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 или дифференциальных уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

В математике рассматривается множество различных типов и видов матриц. Таковы, например, единичная, симметричная, кососимметричная, верхнетреугольная (нижнетреугольная) и т. п. матрицы. Для нас же наиболишьй интерес представляют верхнетреугольные матрицы.

В данной лабораторной работе нам предстоит написать собственную реализацию верхнетреугольных матриц, основанных на векторах. То есть, строками матрицы будут являтся вектора размера n, n-1, n-2 … 1. Для этого нам также потребуется реализовать класс для работы с векторами.

# Постановка задачи

1. Написать класс, реализующий вектор.
2. Написать класс, наследуемый от класса вектор, реализующий верхнетреугольную матрицу.
3. Написать тесты на основе Google Tests для проверки работы классов.

# Руководство пользователя

Чтобы использовать данный класс в своем проекте, необходимо подключить библиотеку «Matrix.h». Библиотека позволяет:

* Создавать объекты типа *матрица* и *вектор:*

TMatrix m1; **конструктор по умолчанию (размер равен нулю)**

TMatrix m2(10); **конструктор с параметром**

TMatrix m3(m2); **конструктор копирования**

TVector v1; **конструктор по умолчанию (размер равен нулю)**

TVector v2(10); **конструктор с параметром**

TVector v3(v2); **конструктор копирования**

* Применять математические операции:

TMatrix m1, m2, m3;

m3 = m1 + m2;

m3 = m2 – m1;

m3 = m1 \* m2;

TVector v1, v2, v3;

v3 = v1 + v2;

v3 = v1 – v2;

v3 = v1 \* v2;

* Получать размер *вектора*:

v1.GetSize();

* Проверять на *пустоту*:

v1.IsEmpty();

* Проверятьна *полноту* и *пустоту* соответственно*:*

v1.IsFull();

v1.IsEmpty();

* Использовать одичночные операторы:

v1[0] = 1;

v1++;

* Проиводить операции над обьектами и числом:

v3 = v1 + 1; **прибавить к каждой компоненте число**

v3 = v1 - 1; **вычесть аналогично**

v3 = v1 \* 2; **умножить**

# Руководство программиста

## Описание структуры программы

Проект состоит из следующих модулей:

1. Vector

Модуль cодержит файл код «main.cpp», в котором продемонстрирован пример использования структуры.

1. VectorLib

Модуль содержит файл заголовок «Vector.h», описывающий структуру «вектор», а также файл кода «main.cpp». Так как класс шаблонный, main содержит лишь подключение.

1. VectorTest

Модуль содержит файлы кода «test\_main.cpp», «VectorTest.cpp». В последнем реализованы тесты для проверки корректности работы методов данного класса.

1. Matrix

Модуль cодержит файл код «main.cpp», в котором продемонстрирован пример использования структуры.

1. MatrixLib

Модуль содержит файл заголовок «Matrix.h», описывающий структуру «матрица», а также файл кода «main.cpp». Так как класс шаблонный, main содержит лишь подключение.

1. MatrixTest

Модуль содержит файлы кода «test\_main.cpp», «MatrixTest.cpp». В последнем реализованы тесты для проверки корректности работы методов данного класса.

## Описание структур данных

#### *Структура TVector*

Структура **TVector**. Поля со спецификатором доступа «protected»:

* + **int n;** *Размер вектора*
  + **T\* v;** *Компоненты вектора*

Поля со спецификатором доступа «public»:

* **TVector(int \_n = 0);** *контсруктор по-умолчанию*
* **TVector(const TVector &other);** *контсруктор копирования*
* **virtual TVector& operator=(const TVector &other);** *оператор =*
* **int GetSize() const { return n; }** *метод, позволяющий**получить размер вектора*
* **T& operator[](int i);** *оператор индексации*
* **virtual TVector operator+()const;** *оператор сложения с числом*
* **TVector operator++();** *инкремент*
* **virtual TVector operator++(int);** *инкремент*
* **virtual TVector operator-()const;** *оператор разности с числом*
* **virtual TVector operator--();** *декремент*
* **virtual TVector operator--(int);** *декремент*
* **virtual TVector operator+(const TVector &other);** *оператор сложения с другим вектором*
* **virtual TVector operator-(const TVector &other);** *оператор разности с другим вектором*
* **virtual TVector operator\*(const TVector &other);** *оператор умножения на другой вектор*
* **virtual TVector operator\*(T ch);** *оператор умножения на число*
* **virtual bool operator==(const TVector &other);** *оператор равенства*
* **virtual bool operator!=(const TVector &other);** *оператор неравенства*
* **virtual ~TVector();** *деструктор*

#### *Структура TVector*

Структура TVector. Поля со спецификатором доступа «protected»:

* **TVector(int \_n = 0);** *конструктор по-умолчанию*
* **TVector(const TVector &other);** *конструктор с параметром*
* **virtual TVector& operator=(const TVector &other);** *оператор* ***=***
* **int GetSize() const { return n; }** *метод, позволюящий получить размер матрицы*
* **T& operator[](int i);** *оператор индексации*
* **virtual TVector operator+()const;** *оператор +*
* **TVector operator++();** *инкремент*
* **virtual TVector operator++(int);** *инкремент*
* **virtual TVector operator-()const;** *оператор -*
* **virtual TVector operator--();** *декремент*
* **virtual TVector operator--(int);** *декремент*
* **virtual TVector operator+(const TVector &other);** *оператор +*
* **virtual TVector operator-(const TVector &other);** *оператор -*
* **virtual TVector operator\*(const TVector &other);** *оператор \**
* **virtual TVector operator\*(T ch);** *оператор \* на число*
* **virtual bool operator==(const TVector &other);** *оператор равенства*
* **virtual bool operator!=(const TVector &other);** *оператор неравенства*
* **virtual ~TVector();** деструктор

## Описание алгоритмов

Для этого конструктор класса TMatrix вызывает конструктор класса TVector, передавая в качестве шаблона <T> – <TVector>. Кроме этого действия для создания матрицы конструктор принимает ее размерность.

Проверка на равенство и неравенство, перегрузка операторов сложения и вычитания.

Для сравнения матриц на равенство необходимо выполнить проверку на равенство вектора векторов. Следовательно достаточно вызвать аналогичные операторы для TVector и вернуть результат их работы (в коде: TVector<TVector<T> >::operator==(A) и TVector<TVector<T> >::operator!=(A)).

Для перегрузки операторов сложения и вычитания требуется сложить/вычесть соответствующие элементы каждой из матриц. Снова на помощь приходит то, что матрица представлена как TVector <TVector>. Когда мы будем складывать элементы строки одной матрицы с элементами строки другой матрицы – мы будем выполнять сложение соответствующих векторов. И так для каждой строки. Следовательно, при выполнении аналогичных операторов для TVector <TVector> будет получен верный результат и для TMatrix (в коде: TVector <TVector<T> > :: operator+(MT) и TVector <TVector<T> > :: operator-(MT)).

# Заключение

В данном лабораторной работе мне удалось:

* Успешно реализовать класс для вектора
* Успешно реализовать класс для матрицы
* Продемонстрировать пример использования данного класса
* Написать тесты на основе Google Tests для проверки корректной работы данного класса

# Литература

Интернет-источники:

1. <https://ru.wikipedia.org/wiki/Структура_данных>
2. [https://ru.wikipedia.org/wiki/Матрица\_(математика)](https://ru.wikipedia.org/wiki/Стек)
3. [https://habr.com/ru/post/277421/](https://habr.com/post/341586/)

Книги:

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
2. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.