Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Матрицы

Отчет по лабораторной работе

Выполнил:

студент ИИТММ гр. 381706-2

Антипин А.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc532933808)

[Постановка целей и задач 4](#_Toc532933809)

[Руководство пользователя 5](#_Toc532933810)

[Руководство программиста 6](#_Toc532933811)

[Описание структуры программы 6](#_Toc532933812)

[Описание структур данных 7](#_Toc532933813)

[Описание алгоритмов 7](#_Toc532933814)

[Эксперименты 10](#_Toc532933815)

[Результаты работы программы 10](#_Toc532933816)

[Время выполнения 11](#_Toc532933817)

[Заключение 13](#_Toc532933818)

[Литература 14](#_Toc532933819)

[Приложение 15](#_Toc532933820)

# Введение

Матрицы являются одним из самых удобных способов представления информации. Они применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. Также они нашли широкое применение в программировании, так как матричный способ представления графической информации является наиболее простым и удобным, если не единственным наиболее правильным.

Операции, определенные над матрицами, также обеспечивают хорошее быстродействие программы и достаточно легко реализуются в программировании. Этими операциями являются: сложение и вычитание матриц одинакового размера, умножение матриц друг на друга и на скаляр.

Помимо матриц общего вида, для которых наиболее естественной и наиболее часто используемой представляется программная реализация в виде двумерного массива, 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 потому более эффективных.

В данной работе будут рассмотрены верхнетреугольные матрицы, которые представляют собой квадратные матрицы, элементы которой ниже главной диагонали равны нулю.

# Постановка целей и задач

Основной целью лабораторной работы является создание программных средств хранения и обработки верхнетреугольных матриц. Средства обработки включают в себя выполнение матричных операций, таких как:

* сложение и вычитание двух матриц
* умножение двух матриц
* умножение матрицы на скаляр

Для реализации алгоритмов будет использоваться 2 шаблонных класса:

* Vector
* Matrix

Для проверки правильности работы этих классов будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая использует данный класс матриц.

# Руководство пользователя

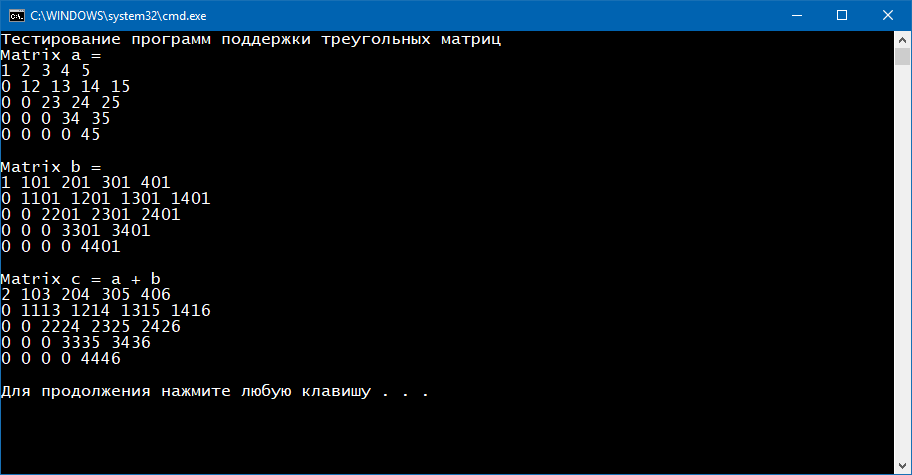
После запуска программы пользователя встречает консольное окно (рис. 1):

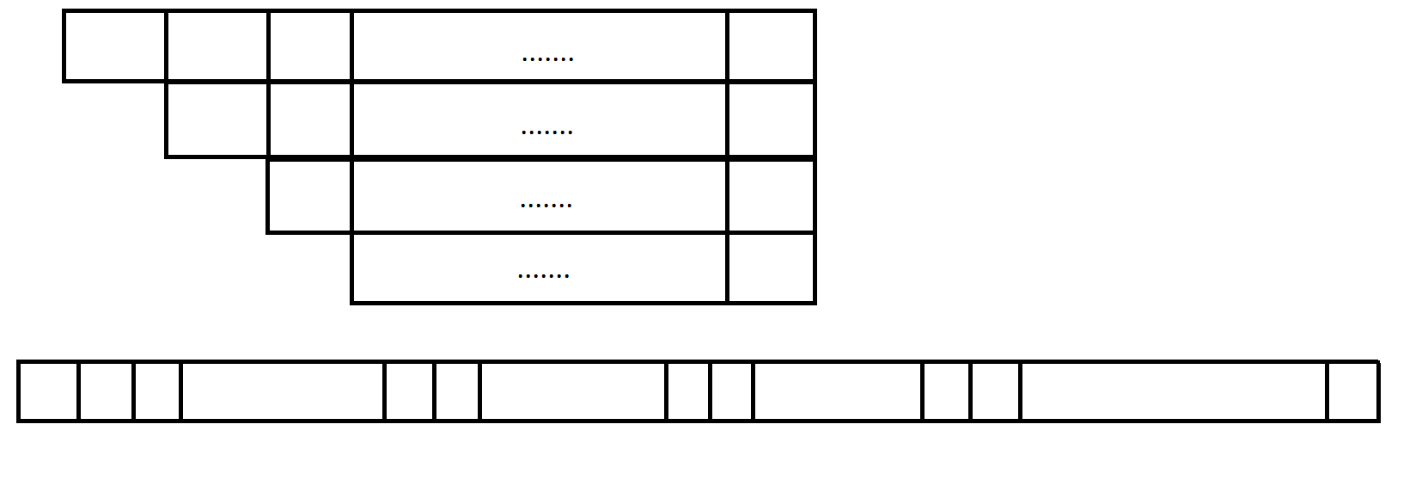
рис. 1 (вывод сложения 2 матриц для пользователя)

в котором выведены две матрицы: a и b, а также матрица c, являющаяся результатом сложения этих двух матриц.

# Руководство программиста

## Описание структуры программы

Матрица будем реализовывать с помощью набора векторов:



То есть для реализации алгоритмов будет использовано 2 класса:

* Класс «Вектор» (Vector).
* Класс «Матрицы» (Matrix), который будет использовать класс векторов.

А также проект использующий фреймворк Google Test, для проверки правильности работы этих классов и тесовый проект, который будет показываться пользователю.

Древо классов

**Класс gtest**

**test\_main.cpp**

**vector\_test.cpp**

**matrix\_test.cpp**

**Проект для пользователя**

**main.cpp**

**matrix.h**

**Класс matrix**

**vector.h**

**Класс vector**

**Класс Vector:**

Класс вектор задает работу с классом matrix посредством возможности создания строк этой матрицы и работы с отдельными элементами этого массива. Также можно выполнять операции, связанные с векторами. Такие как сложение, вычитание векторов, скалярное произведение и др.

**Класс Matrix:**

Класс matrix содержит реализацию работы с верхнетреугольными матрицами. В нем реализованы такие методы, как сложить, вычесть, перемножить 2 матрицы.

**Класс gtest:**

Класс gtest реализует тестирование классов vector и matrix, по средствам фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов. В файле «vector\_test.cpp» реализованы тесты для класса vector, а в файле «matrix\_testcpp» соответственно для класса matrix.

**Проект Vector:**

В данном проекте реализован примет использования верхнетреугольных матриц, который в конечном итоге и будет доступен пользователю.

## Описание структур данных

**Класс Vector:**

template <class T> - шаблон класса T

int l; - длина задаваемого вектора

T\* m; - указатель на первый элемент вектора

int StartIndex; - индекс первого элемента, для удобного индексирования элементов в верхнетреугольных матрицах.

**Класс Matrix:**

template<class A> - шаблон класса А

class Matrix : public Vector<Vector<A>> - наследование от класса Vector другого Vector, т.е. получается «вектор в векторе».

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| Vector<T>::Vector() | Конструктор по умолчанию для класса vector. |
| Vector<T>::Vector(const int \_l, const int \_si) | Конструктор vector, который принимает длину вектора и его стартовый индекс. |
| Vector<T>::Vector(const int \_l) | Конструктор vector, который принимает длину вектора. |
| Vector<T>::Vector(Vector<T> &A) | Конструктор копирования класса vector. |
| Vector<T>::~Vector() | Деструктор класса vector. |
| int Vector<T>::GetSize() | Метод, который возвращает размер текущего вектора. |
| int Vector<T>::GetStartIndex() | Метод, который возвращает стартовый индекс вектора. |
| T& Vector<T>::GetValue (const int n) | Метод, который позволяет получить значение конкретного элемента в векторе, по его индексу с его контролем. |
| T& Vector<T>::operator[]  (const int n) | Перегрузка оператора индексации. |
| Vector<T>& Vector<T>::operator=(const Vector<T>& A) | Перегрузка оператора присваивания векторов. |
| bool Vector<T>::operator==(const Vector<T>& A) | Перегрузка оператора равно, который возвращает «правда», если векторы равны, и «лож», если нет. |
| bool Vector<T>::operator!=(const Vector<T>& A) | Перегрузка оператора равно, который возвращает «лож», если векторы равны, и «правда», если нет. |
| Vector<T> Vector<T>::operator+(const Vector<T>& A) | Перегрузка оператора сложения векторов, к каждому элементу первого вектора прибавляется соответственный элемент второго. |
| Vector<T> Vector<T>::operator-(const Vector<T>& A) | Перегрузка оператора вычитания векторов, из каждого элемента первого вектора вычитается соответственный элемент второго. |
| T Vector<T>::operator\*(const Vector<T>& A) | Перегрузка оператора умножения векторов (скалярного), возвращается элемент типа T. |
| Vector<T> Vector<T>::operator+(const T& A) | Прибавление к каждому элементу вектора константной величины. |
| Vector<T> Vector<T>::operator-(const T& A) | Вычитание из каждого элемента вектора константной величины. |
| Vector<T> Vector<T>::operator\*(const T& A) | Умножение вектора на скаляр. |
| friend std::istream &operator>>(std::istream &in, Vector &a) | Перегрузка ввода вектора. |
| friend std::ostream &operator<<(std::ostream &out, Vector &a) | Перегрузка вывода вектора на консоль. |
| Matrix<A>::Matrix(int \_size) | Конструктор matrix, который принимает размер матрицы. |
| Matrix<A>::Matrix(Matrix <A> &B) : Vector<Vector<A>>(B) | Конструктор копирования матрицы, который наследуется от конструктора копирования вектора с шаблоном Vector<Vector<A>>(B). |
| Matrix<A>::Matrix(Vector<Vector<A>> &B) | Конструктор приведения типов из вектора «векторов» в матрицу. |
| Matrix<A>& Matrix<A>::operator=(Matrix<A>& B) | Перегрузка оператор присваивания для матриц. |
| bool Matrix<A>::operator==(Matrix<A>& B) | Перегрузка оператора равно для матриц. |
| Matrix<A> Matrix<A>::operator+(Matrix<A>& B) | Перегрузка оператора сложения для матриц. |
| Matrix<A> Matrix<A>::operator-(Matrix<A>& B) | Перегрузка оператора вычитания для матриц. |
| Matrix<A> Matrix<A>::operator\*(Matrix<A>& B) | Перегрузка оператора умножения для матриц. |
| Matrix<A> Matrix<A>::operator/(Matrix<A>& B) | Перегрузка оператора деления для матриц (умножение на обратную матрицу) |
| friend std::istream & operator>>(std::istream &in, Matrix &mt) | Перегрузка ввода матрицы. |
| friend std::ostream & operator<<(std::ostream &out, Matrix &mt) | Перегрузка вывода матрицы на консоль. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Оператор присваивания вектора:

* Принимает константную ссылку на объект этого же класса, т.е. vector
* Если не идет попытка присвоить vector самому себе (if (m != A.m)), то поля вектора А присваиваются полям текущего вектора.
* Если в поле m присутствовали элементы, то массив очищается, и выделяется память нужного размера. После в цикле поэлементно присваиваются элементы вектора.
* В конце возвращается указатель на текущий объект.

Умножение матриц:

* Принимает ссылку на объект этого же класса, т.е. matrix.
* Создает матрицу размером l (результирующая матрица).
* В цикле от 0 до l (прохождение по строкам) идет цикл (столбцы матрицы) с i – го элемента (если отталкиваться от обычной матрицы, без представления «вектора векторов») до l, в котором еще 1 цикл от i до g + 1, в котором происходит перемножение элементов матрицы и их запись в результирующую матрицу.
* После всех циклов происходит возврат результирующей матрицы.

# Эксперименты

## Время выполнения

Рассмотрим время выполнения некоторых методов, различающихся свой теоретической сложностью (время приведено в наносекундах):

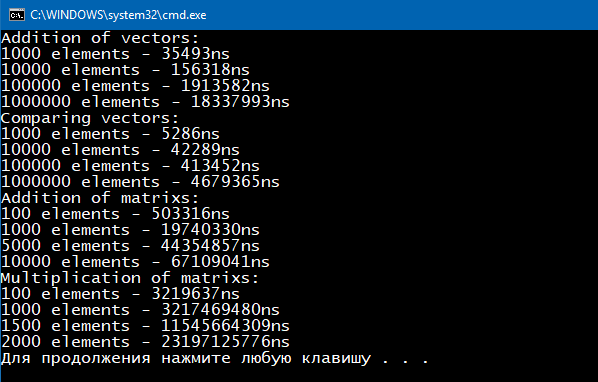
Сложение векторов – теоретическая сложность O(n), сравнение векторов – теоретическая сложность O(n), сложение матриц – теоретическая сложность O(n), умножение матриц – сложность O(n3). На практике получаются такие числа (рис. 2):

рис. 2 (время работы некоторых методов при разном количестве элементов)

Составим диаграмму с временем работы:

С учетом погрешности измерения, можно сказать, что теоретическая сложность алгоритмов совпадает с практическими измерениями.

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# Заключение

В заключении можно сказать, что все поставленные цели и задачи были выполнены, а именно: созданы классы «vector» и «matrix» с реализованными методами сложения, вычитания и умножения матриц, а также написаны к ним тесты, и они успешно пройдены.

# Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.