Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет информационных технологий математики и механики

Отчёт по лабораторной работе

**Структуры хранения матриц специального вида**

Выполнил:

студент ф-та ИТММ гр. 381808-2

Ратов Святослав.

Проверил:

ассистент каф. МО ЭВМ, ВМК

Панов А.

Нижний Новгород

2019 г.

Содержание

[Введение 3](#_Toc22161443)

[Постановка задачи 4](#_Toc22161444)

[Руководство пользователя 5](#_Toc22161445)

[Руководство программиста 6](#_Toc22161446)

[Описание структуры программы 6](#_Toc22161447)

[Описание структур данных 6](#_Toc22161448)

[Описание алгоритмов 7](#_Toc22161449)

[Эксперименты 11](#_Toc22161450)

[Заключение 12](#_Toc22161451)

[Литература 13](#_Toc22161452)

[Приложения 14](#_Toc22161453)

[Приложение 1 14](#_Toc22161454)

[Приложение 2 14](#_Toc22161455)

# Введение

Понятие *Матрица* в европейской науке было введено в работах У. Гамильтона3 и А. Кэли4 в середине XIX века.

Матричные обозначения широко распространены в современной математике и её приложениях. *Матрица* – полезный аппарат для исследования многих задач теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений.

Следствием разнообразия областей применения матричного аппарата в современной науке является наличие в любом из больших математических программных комплексов (Mathcad, Mathematica, Derive, Mapple) подсистем, выполняющих операции над матрицами, а также существование специальных программных библиотек (ScalaPack, PlaPack), рассчитанных на обработку огромных (десятки и сотни тысяч строк) матриц, в том числе с использованием распределенных (параллельных) вычислений.

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, …). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры, и потому более эффективных. Изучению некоторых из них посвящена данная работа.

# Постановка задачи

Реализовать структуру хранения данных, представленную в виде классов TVector и TMatrix, которая позволяет применять к элементам следующие операции:

Для векторов:

* Сравнение;
* Прибавление скаляра;
* Вычитание скаляра
* Вычисление длины.
* Умножение на скаляр
* Сложение векторов
* Вычитание векторов
* Скалярное произведение векторов

Для матриц:

* Умножение на скаляр
* Сравнение
* Умножение на вектор
* Сложение матриц
* Вычитание матриц
* Умножение матриц

Реализовать необходимые тесты с помощью google test.

# Руководство пользователя

Для тестирования созданных классов можно использовать программу верхнетреугольной матрицы, которая выводит в отформатированном виде две верхнетреугольных матрицы и результат сложения этих матриц.

Для подключения библиотеки «staticLib.h» необходимо в рабочий проект добавить проект библиотеки

1. В свойствах проекта указать ссылку на библиотеку
2. Создать включение заголовочного файла «TMatrix.h», указав в свойствах его путь.

# Руководство программиста

## Описание структуры программы

Программа содержит в себе следующие проекты:

1. Vector.h, Vector.cpp – модуль, реализующий структуру данных *Вектор*;
2. Matrix.h, Matrix.cpp – модуль, реализующий структуру данных *Матрица*;
3. Main.cpp – модуль программы тестирования.
4. test\_matrix – набор тестов для тестирования классов TMatrix, TVector, реализованный с помощью библиотеки для модульного тестирования **Google C++ Testing Framework**.

## Описание структур данных

Матрица представляет из себя класс utmatrix. Данный класс позволяет создать вектор и матрицу с элементами типа ValType, являющегося пользовательским типом данных.

Класс включает в себя следующие поля:

* ValType\* pVector; // динамически выделяемый непрерывный участок памяти
* int Size; // размер вектора
* int StartIndex; // индекс первого элемента вектора

Класс позволяет работать со следующими методами и перегрузками:

* int GetSize() { return Size; } // размер вектора
* int GetStartIndex() { return StartIndex; } // индекс первого элемента
* ValType& operator[](int pos); // доступ
* bool operator==(const TVector& v) const; // сравнение
* bool operator!=(const TVector& v) const; // сравнение
* TVector& operator=(const TVector& v); // присваивание

// скалярные операции

* TVector operator+(const ValType& val); // прибавить скаляр
* TVector operator-(const ValType& val); // вычесть скаляр
* TVector operator\*(const ValType& val); // умножить на скаляр

// векторные операции

* TVector operator+(const TVector& v); // сложение
* TVector operator-(const TVector& v); // вычитание

## ValType operator\*(const TVector& v); // скалярное произведение

// ввод-вывод

* friend istream& operator>>(istream& in, TVector& v)

friend ostream& operator<<(ostream& out, const TVector& v)

Для матрицы:

* TMatrix(int s = 10);
* TMatrix(const TMatrix& mt); // копирование
* TMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа
* bool operator==(const TMatrix& mt) const; // сравнение
* bool operator!=(const TMatrix& mt) const; // сравнение
* TMatrix& operator= (const TMatrix& mt); // присваивание
* TMatrix operator+ (const TMatrix& mt); // сложение
* TMatrix operator- (const TMatrix& mt); // вычитание
* friend istream& operator>>(istream& in, TMatrix& mt) // ввод / вывод
* friend ostream& operator<<(ostream& out, const TMatrix& mt)

## Описание алгоритмов

**Класс TVector:**

1. Конструктор



Сначала идет проверка на число входящее в диапозон от 0 до MAX\_VECTOR\_SIZE. Выделяется память для пустого вектора размера size.

1. Конструктор копирования



В Size и StartIndex присваиваем значения полей v, затем выделяем необходимое количество новой памяти . В цикле происходит поэлементное копирование данных.

1. Деструктор



Если есть элементы, то происходит удаление вектора, затем присваиваивается Size и StartIndex значение 0.

1. Доступ



Проверка на вход в диапазон массива. Возвращает нужный элемент.

1. Сравнение



Происходит сравнение по Size и по StartIndex. Затем происходит сравнение поэлементно.

1. Присваивание



Сначала выполняется проверка на равенство векторов. Если размеры векторов не равны, то создается новый вектор с размером второго вектора. Старый вектор удаляется. Затем идет поэлементное присваивание элементов.

1. Прибавить скаляр



Создается временный буффер для хранения указателя на текущий вектор. Происходит поэлементное прибавление на значение.

1. Вычесть скаляр



Создается временный буффер для хранения указателя на текущий вектор. Происходит поэлементное вычитание на значение.

1. Умножить на скаляр



Создается временный буффер для хранения указателя на текущий вектор. Происходит поэлементное умножение на значение.

1. Сложение



Идет проверка на размер векторов. Создается временный буффер для хранения указателя на текущий вектор. Происходит поэлементное сложение второго вектора к первому.

1. Вычитание

  
Идет проверка на размер векторов. Создается временный буффер для хранения указателя на текущий вектор. Происходит поэлементное вычитание второго вектора из первого.

1. Скалярное произведение

  
Аналогичная реализация (11).

**Класс TMatrix:**

1. Конструктор



Сначала идет проверка на вход в диапазон. Создатся треугольная матрица.

1. Сравнение



Аналогично (5) для вектора.

1. Присваивание



Аналогично (6) для вектора.

1. Сложение



Аналогично (10) для вектора

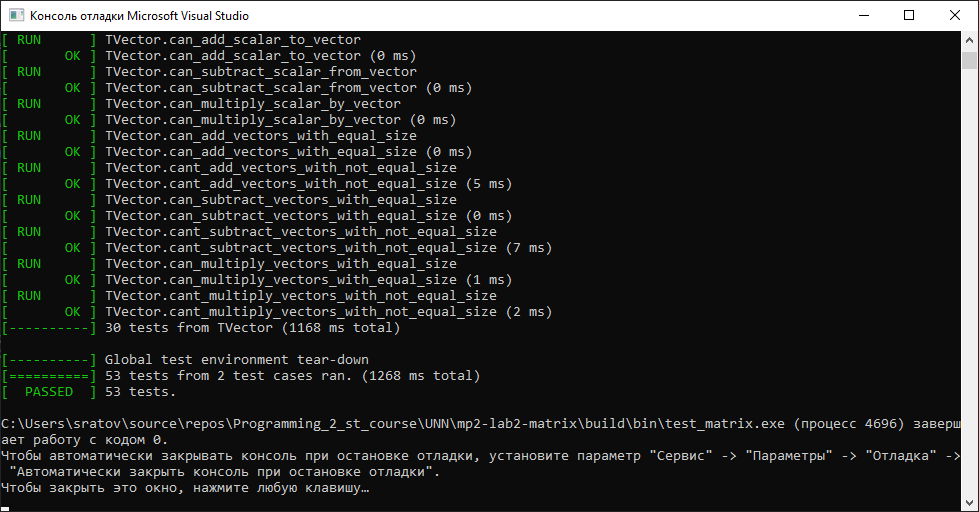
1. Вычитание



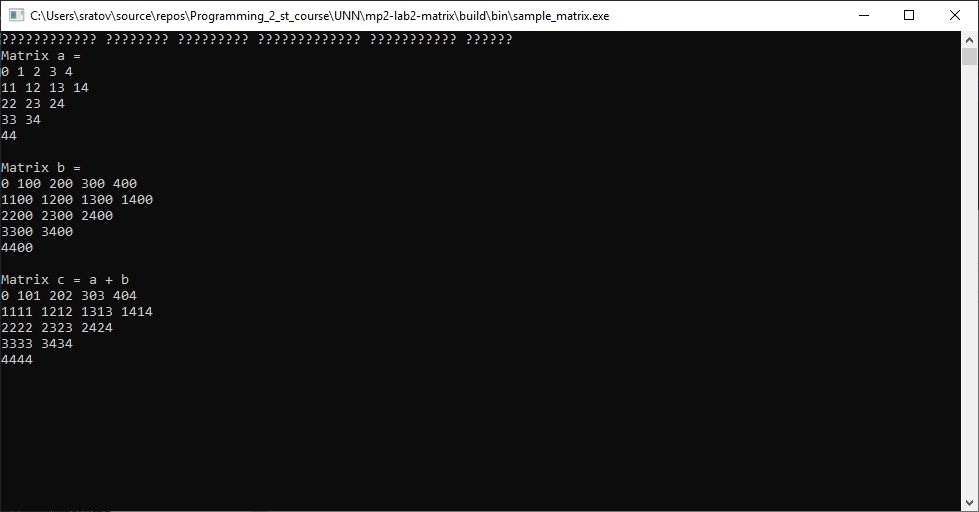
Аналогично (11) для вектора

# Эксперименты

1. Программа проходит все тесты Google



1. Программа выводит правильную треугольную матрицу и имеет форматированный вывод



# Заключение

Приобретены навыки unit-тестирования на основе GoogleTests

# Литература

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
9. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
10. Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.