Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики и механики

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Отчёт по лабораторной работе

**Реализация классов для работы с матрицами и векторами.**

Выполнил:

студент ИИТММ гр. 381906-2

Кулемин П. А.

Проверил:

программист лаб. суперкомпьютерных технологий

и высокопроизводительных

вычислений каф. математического обеспечения

и суперкомпьютерных технологий

Усова М. А.

Нижний Новгород

2019 г.

Содержание

I. [Введение 3](#_Toc24457830)

II. [Постановка задачи](#_Toc24457831) 4

III. [Руководство пользователя](#_Toc24457832) 5

IV. [Руководство программиста](#_Toc24457833) 6

1. [Описание структуры программы](#_Toc24457834) 6

2. [Описание структур данных](#_Toc24457835) 7

3. [Описание алгоритмов](#_Toc24457836) 7

V. [Эксперименты](#_Toc24457837) 10

VI. [Заключение](#_Toc24457838) 11

VII. [Литература](#_Toc24457839) 12

VIII. [Приложение 1](#_Toc24457840)3

# 1. Введение

Матрицы и векторы, это важная часть векторной алгебры, и поэтому информатизация и автоматизация в этой области математики крайне важна. Именно для этого мной были созданы классы для работы с векторами и матрицами, а также произведена теоретическая и практическая оценка времени работы различных арифметических операций над матрицами.

# 2. Постановка задачи

Реализовать классы для работы с векторами и матрицами использовать шаблоны. Продемонстрировать их работу на примере.

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 или скаляр), матриц (матрица +, -, \* матрица или скаляр).

Оценить асимптотически время работы матричных и векторных операций \* сравнить оценку с реальным временем работы. Сделать вывода.

# 3. Руководство пользователя

Шаги, совершаемые пользователем:

1. Создать объект типа Matrix или Vector, в зависимости от необходимости.
2. Расширить матрицу или вектор на нужное количество элементов, используя конструкторы либо.
3. Произвести необходимые математические операции.

# 4. Руководство программиста

## 4.1. Описание структуры программы

Программа состоит из 6 решений.

В решении ALL\_BUILD, gtest, а также ZERO\_CHECK определен вспомогательный код, необходимый для работы тестовой системы и всей программы в целом.

В решении matrix содержится 1 модуль umatrix.h. В нем содержится объявление, а также определение шаблонного класса TVector, и его наследника TMatrix.

В решении sample\_matrix содержится один модуль, sample\_matrix.cpp содержащий в себе обьявление функции main с примером работы над объектами класса TVector и TMatrix.

В решении test\_matrix содержится 3 модуля: test\_main.cpp, test\_tvector.cpp, test\_tmatrix.cpp. В них содержится набор тестов, с помощью которых мы подвергаем нашу программу проверке на корректную работу.

## Описание структур данных

В программе определены два следующих шаблонных класса:

Class TMatrix

Class TVector

Внутри класса TVector определены следующие protected поля:

* ValType\* pvector– шаблонный указатель.
* Int size– целочисленная длина вектора.
* Int StartIndex– целочисленный номер стартовой ячейки для данного вектора.

Внутри класса Matrix определен следующий набор public-методов

(плюс конструкторы и деструктор):

* TVector(int s = 10, int si = 0) – конструктор по умолчанию, и инциализатор в одномб принимает длину вектора и стартовый индекс.
* TVector(const TVector& v) – конструктор копирования, принимает на вход объект типа TVector, создает объект с теми же характеристиками что и переданный, и массивом того же содержимого.
* Virtual ~TVector() – виртуальный деструктор, очищает выделенную конструкторами память.
* Int GetSize () – метод возвращающий длину вектора.
* Int GetStartIndex () – метод возвращающий стартовый индекс вектора.
* Void SetSize(int) – метод устанавливающий размер вектора в соответствии с переданным аргументом.
* Void SetStartIndex(int)- метод устанавливающий стартовый индекс в соответствии с переданным аргументом.
* VaType\*\* GetVector () – метод возвращающий указатель на блок памяти с вектором.
* ValType& operator[](int) –перегрузка оператора индексации.
* ValType& operator [ ](int) const - константная перегрузка оператора индексации.
* Bool operator == (Const TVector&) – перегрузка оператора равно.
* Bool operator != (Const TVector&) – перегрузка оператора неравно.
* TVector& operator = (TVector&) – перегрузка оператора присваивания.
* TVector operator +(const ValType& ) – перегрузка оператора сложения для объектов вектор и скаляр. Математически определяется как вектор с координатами, каждая из которых увеличена на скаляр.
* TVector operator -(const ValType&) - перегрузка оператора вычитания, по своей сути аналогична перегрузки сложения.
* TVector operator \*(const ValType&) - перегрузка оператора умножения на скаляр.
* TVector operator +(const TVector&) - перегрузка оператора сложения для двух векторов.
* TVector operator -(const TVector&) - перегрузка оператора вычитания, для двух векторов.
* ValType operator \*(const ValType&) - перегрузка оператора умножения, для двух векторов
* Int FindIndex() – возвращает номер первой ненулевой координаты вектора, или -1
* friend ostream& operator <<(ostream& out, const TVector& rhs) – перегрузка оператора вывода, позволяет выводить вектора на экран
* friend istream& operator >>(istream& in, TVector& rhs) - перегрузка оператора ввода, позволяет вводить вектора уже заданного размера, либо задавать размер и вводить самостоятельно.

Шаблонный класс TMatrix, является public, наследником класса TVector, при этом параметр шаблона определен как TVector<ValType> > поэтому никаких полей в этом классе не определено. Также почти 70% методов, он наследует от класса-родителя, поэтому внутри него, определены только специфичные для него методы:

* TMatrix(int s=10) –конструктор по умолчанию, и конструктор инциализатор .
* TMatrix(const TMatrix&) – конструктор копирования.
* TMatrix(TVector<TVector<ValType> >&&) – move конструктор с параметром.
* TMatrix(TVector<TVector<ValType> >&) конструктор с параметром вектор с шаблоном вектор от шаблона.
* Bool operator ==(const TMatrix&)const – перегрузка оператора равно.
* Bool operator !=(const TMatrix&)const – перегрузка оператора неравно.
* TMatrix& operator =(const TMatrix&) – перегрузка оператора присваивания.
* TMatrix operator +(const TMatrix&) – перегрузка оператора суммы
* TMatrix operator -(const TMatrix&) - перегрузка оператора разности
* friend ostream& operator <<(ostream&, TMatrix&) – перегрузка оператора вывода н экран.
* friend istream& operator >>(istream&, TMatrix&) - перегрузка оператора ввода, с консоли.

## Описание алгоритмов

* Алгоритм суммирования или вычитания матриц:

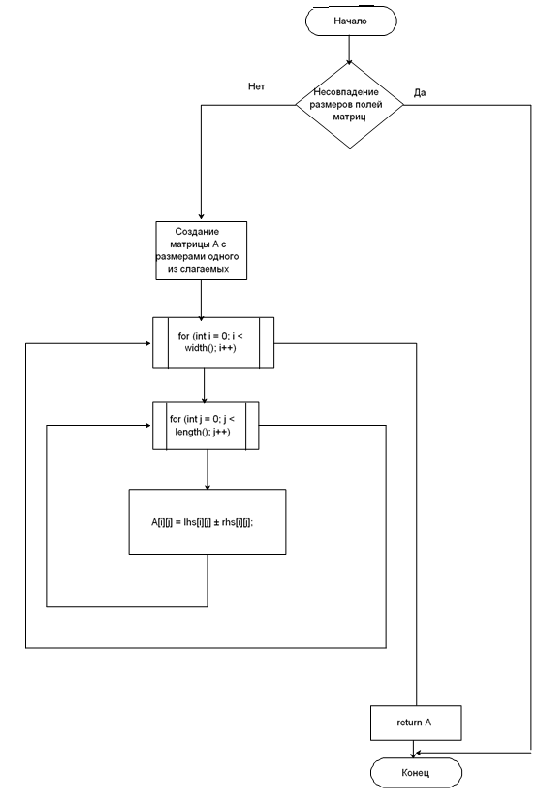


Рисунок 1: Блок-схема алгоритма сложения/вычитания векторов

Поскольку класс TMatrix это публичный наследник класса TVector, то для него нет смысла описывать каким-либо образом арифметическую операцию сложения, так как эта операция в числе других, наследуется от класса TVector.

* Алгоритм умножения двух векторов:

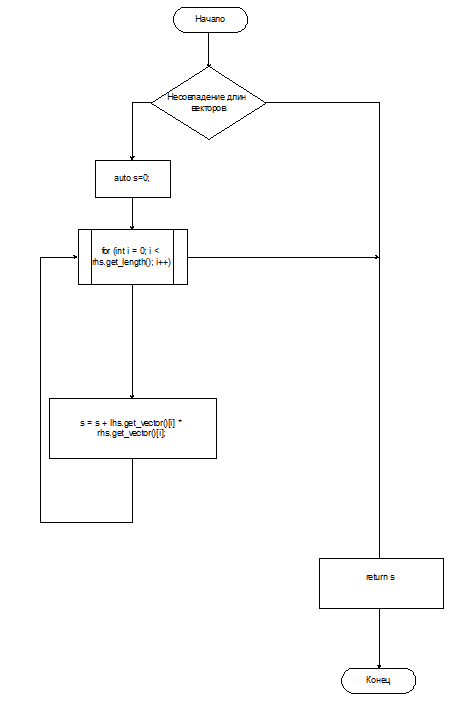


Рисунок 2: Блок-схема умножения двух векторов.

# Эксперименты

Оценим время, которое занимают векторные и матричные операции, с помощью асимптотической сложности. Рассмотрим код отвечающий за суммирование матриц:

TMatrix<ValType>Res(\*this);

for (int i = Res.GetStartIndex(); i < Res.GetSize(); i++)

{

Res[i] = (\*this)[i] + mt[i];

}

Для векторов операция суммы и присваивания линейная значит асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения квадратных матриц (чтобы точнее соотнести результаты замеров с асимптотической сложностью):

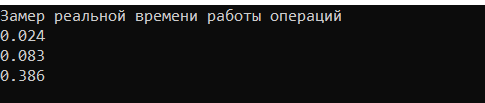


Рисунок 3: Замеры времени, полученные при выполнении сложения матриц (сек.).

На основе полученных данных, построим таблицу соотношения размеров матриц и времени их суммирования.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 2000 | 0.024 |
| 4000 | 0.083 |
| 8000 | 0.386 |

Таблица 1: Результаты замеров времени суммирования матриц.

Теперь оценим время работы векторного умножения с помощью асимптотической сложности. Рассмотрим код отвечающий за умножения вектора на вектор:

if (this->GetSize() != v.GetSize()) throw logic\_error("");

float res=0;

for (int i = 0; i < v.Size - v.StartIndex; i++)

{

res = res + v.pVector[i]\*this->pVector[i];

}

Для элементов вектора операция умножения работает за константу значит асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения квадратных матриц (чтобы точнее соотнести результаты замеров с асимптотической сложностью):



Рисунок 4: Замеры времени, полученные при выполнении умножения (сек.).

На основе полученных данных, построим таблицу соотношения размеров векторов и времени их умножения:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000000 | 0.004 |
| 2000000 | 0.006 |
| 3000000 | 0.012 |

Таблица 1: Результаты замеров времени умножения векторов.

# 6. Заключение

По итогу проведения данной работы, нам удалось реализовать удобные объекты для работы с матрицами и векторами. Также по проведенным экспериментам можно сделать вывод, что с некоторыми погрешностями, затраченное на реализацию матричных, векторных операций время, совпадает с ожидаемым относительно асимптотической сложности.

# 7. Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. <https://ru.wikipedia.org>
3. <https://ru.stackoverflow.com>
4. <https://learnc.info/c/memory_allocation.html>

# 8. Приложение