МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механик**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«ВЕРХНЕТРЕУГОЛЬНЫЕ МАТРИЦЫ»**

**Выполнил(а):** студент(ка) группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Суворов Д.И. /

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc150303615)

[1 Постановка задачи 5](#_Toc150303616)

[2 Руководство пользователя 6](#_Toc150303617)

[2.1 Приложение для демонстрации работы векторов 6](#_Toc150303618)

[2.2 Приложение для демонстрации работы множеств 8](#_Toc150303619)

[3 Руководство программиста 11](#_Toc150303620)

[3.1 Алгоритмы 11](#_Toc150303621)

[3.1.1 Вектор 11](#_Toc150303622)

[3.1.2 Верхнетреугольные матрицы 11](#_Toc150303623)

[3.2 Описание классов 12](#_Toc150303624)

[3.2.1 Класс TVector 12](#_Toc150303625)

[3.2.2 Класс TMatrix 15](#_Toc150303626)

[Заключение 18](#_Toc150303627)

[Литературы 19](#_Toc150303628)

[Приложения 20](#_Toc150303629)

[Приложение А. Реализация класса TVector 20](#_Toc150303630)

[Приложение Б. Реализация класса TMatrix 22](#_Toc150303631)

[Приложение В. Использование функционала векторов 23](#_Toc150303632)

[Приложение Г. Использование функционала верхнетреугольных матриц 24](#_Toc150303633)

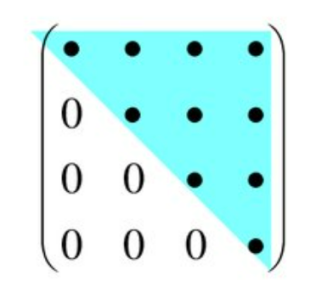
# Введение

Вектор – это математический объект, представляющие собой упорядоченный набор чисел, которые называются компонентами или координатами вектора. Для векторов в линейном пространстве определены операции сложения и разности двух векторов, умножения вектора на число. В евклидовом пространстве определена операция скалярного произведения двух векторов.

Вектора используются во многих областях современной науки. В линейной алгебре вектор – это одно из основополагающих понятий. Вектора образуют различные пространства, описывают системы линейных уравнений. Физика изучает многие векторные величины, то есть такие величины, которые имеют не только численную меру, но и направление. В частности, в механике векторы позволяют описывать движения тел, что позволяет решать задачи, которые механика ставит перед собой. В информационных технологиях векторы используются в качестве структур хранения данных. В компьютерной графике векторы позволяют визуализировать трехмерное пространство. В теории информации упорядоченный набор значений удобен для некоторых алгоритмов кодирования данных и для представления кодовых слов. Таким образом, векторы применяются во многих областях точных наук, это очень удобный инструмент моделирования, инструмент хранения данных. Векторы помогают решать различные задачи.

Ещё одно применение векторов – это представление матриц. Матрица – это прямоугольная таблица определённого размера, содержащая действительные числа в качестве элементов. Один из способов представления матриц является упорядоченный набор векторов.

В частности одним из видов матриц, является верхнетреугольная матрица. Это квадратная матрица (такая матрица, в которой совпадает количество столбцов и строк), в которой все элементы ниже главной диагонали (с левого верхнего до правого нижнего угла) равны 0.



Такой вид матриц интересен для программирования, так как хранить в памяти необходимо только половину матрицы, что, как следствие, вдвое сокращает затраты по памяти, по сравнению с полной матрицей. В том числе такой вид матриц подходит для представления симметричных матриц, так как в них элементы над главной диагональю симметрично равны элементам под главной диагональю. Значит, с целью оптимизации, хранить симметричные матрицы возможно в виде верхнетреугольных, в которых в памяти сохранены только элементы над верхней диагональю.

Симметричные квадратные матрицы используются в линейной алгебре, компьютерной графике, криптографии, во многих разделах физики и в статистике.

Всё вышеперечисленное доказывает активную применимость и высокую важность векторов и матриц в многих областях науки и техники.

# Постановка задачи

Цель – получить практические навыки применения структуры данных вектора и верхнетреугольной матрицы.

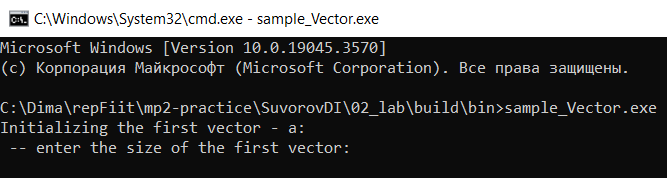
Задачи:

1. Разработать класс, представляющий вектор.
2. Реализовать все необходимые операции с векторами.
3. Разработать класс верхнетреугольной матрицы.
4. Реализовать все необходимые операции с данными матрицами.
5. Протестировать корректность работы разработанного функционала.
6. Сделать вывод о проделанной работе.

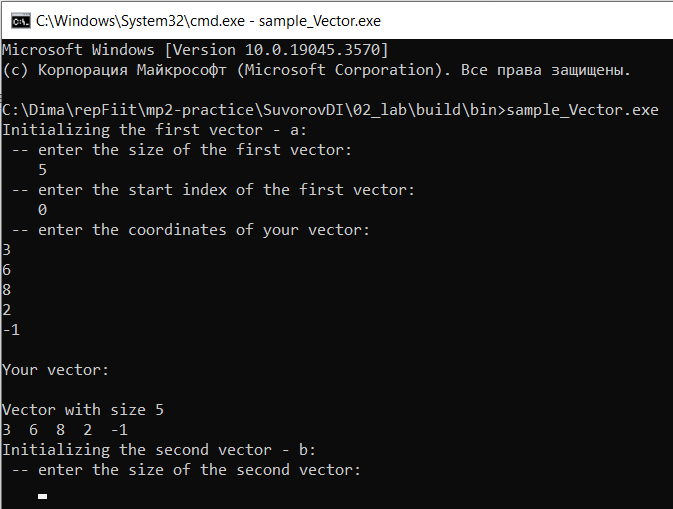
# Руководство пользователя

## Приложение для демонстрации работы векторов

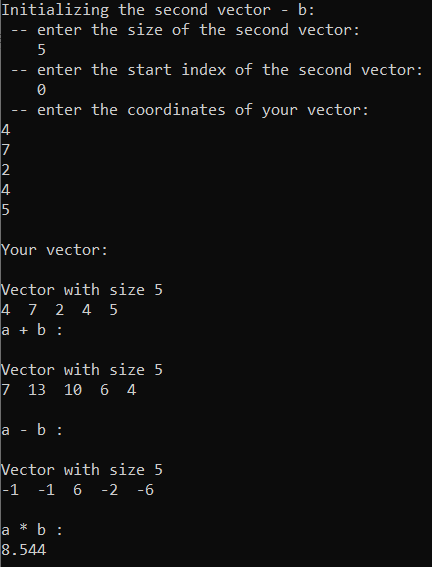
1. Запустить sample\_Vector.exe. В результате появится следующее окно (рис. 1):



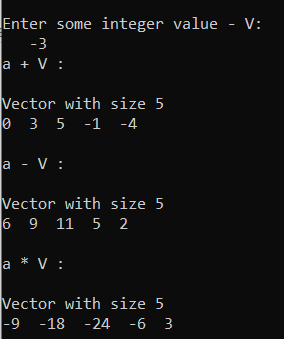
1. Стартовое окно приложения
2. Приложение потребует создать первый вектор «а». Для этого приложение ожидает ввода желаемого размера для вектора «а». После ввода размера приложение будет ожидать ввода стартового индекса вектора (Это значение от 0 до длины вектора не включительно). Затем приложение будет ожидать ввода координат вектора через Enter. После ввода желаемых значений состояние приложения будет следующим (рис. 2).



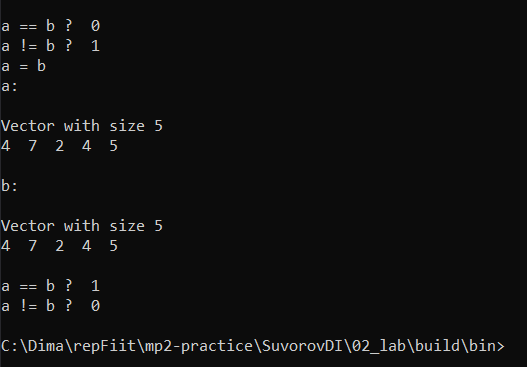
1. Состояние после создания и заполнения первого вектора
2. После того, как вектор «а» был создан и задан приложения ожидает всех тех же действий над вектором «b». После успешного ввода необходимых параметров для вектора «b» приложение выведет результаты векторных операций (сложение векторов, вычитание векторов, скалярное произведение векторов) (рис. 3).



1. Результаты векторных операций
2. После вывода результатов векторных операций приложение ожидает ввода некоторого числового значения, после чего на экране приложения выведутся результаты векторно-скалярных операций (сложение, вычитание, умножение вектора с числом) (рис. 4).



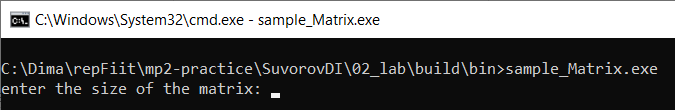
1. Результаты векторно-скалярных операций
2. После вывода результатов скалярно-векторных операций приложение выведет результаты операций сравнений (рис. 5).



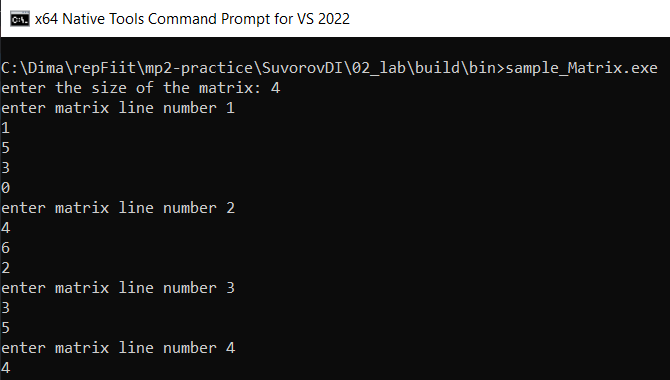
1. Результат операций сравнения – итоговое состояние приложения

## Приложение для демонстрации работы множеств

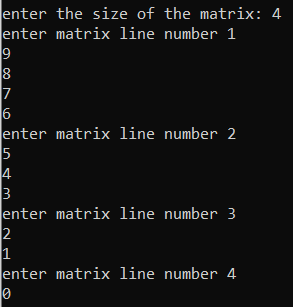
1. Запустить sample\_Matrix.exe. В результате появится следующее окно (рис. 6).



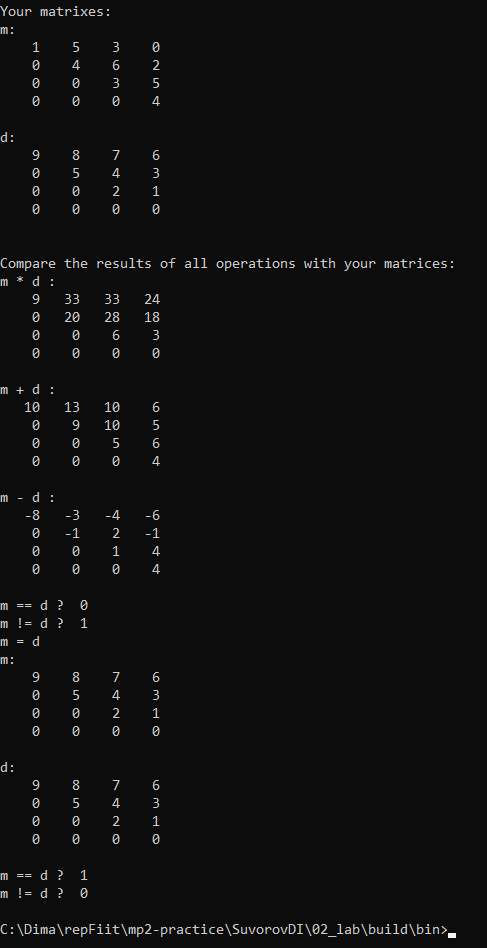
1. Стартовое окно приложения
2. Приложение ожидает ввода необходимых значений для первой верхнетреугольной матрицы, а именно: размер и значения всех элементов   
   (рис. 7).



1. Ввод длины и элементов первой матрицы
2. После ввода необходимых данных о первой матрице программа ожидает ввода данных о второй матрице в аналогичной форме (рис. 8).



1. Данные второй матрицы
2. После чего приложение выведет на экран заданные пользователем верхнетреугольные матрицы, а также результаты матричных операций сложения, вычитания, умножения, а также операций сравнения (рис. 9).



1. Результаты операций над матрицами – итоговое окно приложения

# Руководство программиста

## Алгоритмы

### Вектор

Вектор представляет собой упорядоченный набор элементов. У вектора есть два параметра: длина и стартовый индекс. Размер вектора определяет количество элементов в нём. Стартовый индекс необходим для удобного использования вектора внутри верхнетреугольной матрицы и представляет собой целое число, которое равно номеру строки матрицы, которую представляет собой этот вектор (это число принимает значение от 0 до значения размера матрицы не включительно).

Вектор предоставляет доступ к любому его элементы по номеру. Элементы номеруются с 0 до size-1, где size – размер вектора.

Над векторами определены векторные операции: сложение, вычитание, скалярное произведение. Сложение и вычитание производятся поэлементно и требуют одинаковых размеров от векторов, над которыми они будут выполняться. Скалярное произведение – это сумма произведений соответствующих по номеру элементов двух векторов. Для выполнения скалярного произведения необходимым условием является равенство размеров двух векторов. Для совместимости использования векторов с верхнетреугольными матрицами, не допускается применение векторных операций к векторам с разными стартовыми индексами.

Для векторов определены скалярно-векторные операции: сложение, вычитание, умножение вектора со скаляром (числовым значением). В зависимости от операции результирующим вектором является вектор, каждый элемент которого получен из соответствующего элемента изначального вектора с добавлением, вычитанием, умножением его на указанный скаляр.

Вектора поддерживают сравнение. Равны вектора, если у них равны размер, стартовый индекс и соответствующие элементы. В противном случае – вектора не равны.

Пример выполнения операций над векторами:

а:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 32 | 5 | 0 | 3 | 55 | 33 | -2 | 9 |

б:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | -4 | 10 | 46 | 0 | -1 | 2 | 13 | -5 |

а + б:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 28 | 15 | 46 | 3 | 54 | 35 | 11 | 4 |

а - б:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -4 | 36 | -5 | -46 | 3 | 56 | 31 | -15 | 14 |

а \* б:

### Верхнетреугольные матрицы

Верхнетреугольные матрицы – это квадратные матрицы (такие матрицы, у которых количество столбцов и строк равны), у которых элементы «ниже» главной диагонали равны 0. Такая матрица представляет собой вектор, элементами которого являются векторы, длина которых равны (n – i), а стартовый индекс равен i, где n – это размер матрицы (количество строк и столбцов), i – это номер соответствующего вектора-строки матрицы. Таким образом в памяти не будут храниться нули ниже главной диагонали, что делает хранение такого типа матрицы эффективным с точки зрения затрат по памяти.

Стартовый индекс нужен, чтобы определять принадлежность вектора-строки к матрице определенного размера и его положение в ней и, как следствие, для различия от векторов такой же длины, но расположенных в матрице другого размера на другой позиции внутри неё. Например, в приведенном ниже примере вторая строка первой матрицы не будет равна первой строке второй матрицы, так как в первой матрице вторая строка на самом деле длины 4, и первый элемент её – 0, но в памяти элементы ниже главной диагонали не хранятся, так что отличать между собой эти две похожие строки помогает параметр стартового индекса. Также при выполнении матричных операций над верхнетреугольной матрицей позволит определять настоящий номер столбца элементов ниже первой строки (с учётом не хранящихся в векторах нулей ниже главной диагонали).

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 5 | 5 | 6 |
| 0 | 3 | 4 | 8 |
| 0 | 0 | 9 | 7 |
| 0 | 0 | 0 | 6 |

|  |  |  |
| --- | --- | --- |
| 3 | 4 | 8 |
| 0 | 9 | 7 |
| 0 | 0 | 6 |

Представленные матрицы поддерживают классические матричные операции сложения, вычитания, умножения, которые вводит линейная алгебра для математической модели матрица. При сложении и вычитании двух матриц элемент результирующей матрицы, расположенный на пересечении i-ого столбца и j-ой строки, равны сумме элементов исходных матриц, расположенных на пересечении i-ого столбца и j-ой строки. В матрице, получившейся в результате произведения двух матриц, элемент на пересечении i-ого столбца и j-ой строки равен

(где, n – размер квадратной матрицы; – элемент первой исходной матрицы, расположенный на пересечении i-ого столбца и k-ой строки; – элемент второй исходной матрицы, расположенный на пересечении k-ого столбца и j-ой строки)

Также матрицы поддерживают операцию сравнения. Две матрицы равны, если равны из размеры и соответствующие элементы, в противном случае матрицы считаются не равными.

А:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 5 | 5 | 6 |
| 0 | 3 | 4 | 8 |
| 0 | 0 | 9 | 7 |
| 0 | 0 | 0 | 6 |

Б:

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | -10 | 3 | 8 |
| 0 | 3 | -1 | -9 |
| 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | -12 |

А + Б:

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | -5 | 8 | 14 |
| 0 | 0 | 3 | -1 |
| 0 | 0 | 9 | 9 |
| 0 | 0 | 0 | -6 |

А - Б:

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | -5 | 8 | 14 |
| 0 | 0 | 3 | -1 |
| 0 | 0 | 9 | 9 |
| 0 | 0 | 0 | -6 |

А \* Б:

|  |  |  |  |
| --- | --- | --- | --- |
| 24 | -25 | 7 | -75 |
| 0 | 9 | -3 | -115 |
| 0 | 0 | 0 | -66 |
| 0 | 0 | 0 | -72 |

## Описание классов

### Класс TVector

Объявление класса:

template <typename ValueType>

class TVector {

protected:

  int size;

  int startIndex;

  ValueType\* pVector;

public:

  TVector(int size = 5, int startIndex = 0);

  TVector(const TVector<ValueType>& v);

  ~TVector();

  int GetSize() const;

  int GetStartIndex() const;

  ValueType& TVector<ValueType>::operator[] (const int i);

  int operator== (const TVector<ValueType>& v) const;

  int operator!= (const TVector<ValueType>& v) const;

  const TVector& operator= (const TVector<ValueType>& v);

  TVector operator+ (const ValueType val);

  TVector operator- (const ValueType val);

  TVector operator\* (const ValueType val);

  TVector operator+ (const TVector<ValueType>& v);

  TVector operator- (const TVector<ValueType>& v);

  double operator\* (const TVector<ValueType>& v);

  friend std::istream& operator>> (std::istream& in, TVector<ValueType>& v);

  friend std::ostream& operator<< (std::ostream& out, TVector<ValueType>& v);

};

Поля:

size – размер вектора.

startIndex – стартовый индекс вектора.

pVector – указатель типа ValueType на первый элемент вектора.

Методы:

int GetSize() const;

Назначение: получение пользователем размера вектора.

Выходные данные: целое число – размер вектора.

int GetStartIndex() const;

Назначение: получение стартового индекса вектора.

Выходные данные: целое число – стартовый индекс вектора.

*ValueType*& TVector<*ValueType*>::operator[] (const int *i*);

Назначение: получить элемент вектора под конкретным номером.

Входные данные: номер элемента (индекс) в векторе (целое число от 0 до значения размера вектора не включительно).

Выходные данные: значение элемента вектора под переданным номером.

int operator== (const TVector<*ValueType*>& *v*) const;

Назначение: сравнение двух векторов.

Входные данные: вектор, с которым необходимо сравнить текущий.

Выходные данные: результат сравнения (1, если равны; 0, если не равны).

int operator!= (const TVector<*ValueType*>& *v*) const;

Назначение: сравнение двух векторов.

Входные данные: вектор, с которым необходимо сравнить текущий.

Выходные данные: результат сравнения (0, если равны; 1, если не равны).

const *TVector*& operator= (const TVector<*ValueType*>& *v*);

Назначение: присвоение (копирование) переданного вектора в текущий.

Входные данные: вектор, который будет скопирован.

Выходные данные: результат копирования – ссылка на копию копируемого вектора.

*TVector* operator+ (const *ValueType* *val*);

Назначение: сложение вектора со скаляром.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат сложения.

*TVector* operator- (const *ValueType* *val*);

Назначение: вычитание из вектора скаляра.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат вычитания.

*TVector* operator\* (const *ValueType* *val*);

Назначение: умножения вектора на скаляр.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат умножения.

*TVector* operator+ (const TVector<*ValueType*>& *v*);

Назначение: сложение двух векторов.

Входные данные: вектор.

Выходные данные: результирующий вектор – результат сложения исходного и переданного вектора.

*TVector* operator- (const TVector<*ValueType*>& *v*);

Назначение: вычитание двух векторов.

Входные данные: вектор.

Выходные данные: результирующий вектор – результат вычитания исходного вектора и переданного.

double operator\* (const TVector<*ValueType*>& *v*);

Назначение: вычисление скалярного произведения векторов.

Входные данные: вектор.

Выходные данные: вещественное число – результат скалярного произведения двух векторов.

friend std::*istream*& operator>> (std::*istream*& *in*, TVector<*ValueType*>& *v*);

Назначение: ввод вектора.

Входные данные: ссылка на поток ввода и вектор.

Выходные данные: ссылка на поток ввода.

friend std::*ostream*& operator<< (std::*ostream*& *out*, *TVector*<*ValueType*>& *v*);

Назначение: вывод вектора.

Входные данные: ссылка на поток вывода и вектор.

Выходные данные: ссылка на поток вывода.

### Класс TMatrix

template <typename *ValueType*>

class TMatrix : public TVector<TVector<*ValueType*>> {

  using TVector<TVector<*ValueType*>>::pVector;

  using TVector<TVector<*ValueType*>>::size;

  using TVector<TVector<*ValueType*>>::startIndex;

public:

  TMatrix(int *n* = 5);

  TMatrix(const TMatrix<*ValueType*>& *m*);

  TMatrix(const TVector<TVector<*ValueType*>>& *v*);

  int operator== (const TMatrix<*ValueType*>& *m*) const;

  int operator!= (const TMatrix<*ValueType*>& *m*) const;

  const *TMatrix*& operator= (const TMatrix<*ValueType*>& *m*);

*TMatrix* operator+ (const TMatrix<*ValueType*>& *m*);

*TMatrix* operator- (const TMatrix<*ValueType*>& *m*);

*TMatrix* operator\* (const TMatrix<*ValueType*>& *m*);

  friend std::*istream*& operator>> (std::*istream*& *in*, TMatrix<*ValueType*>& *m*);

  friend std::*ostream*& operator<< (std::*ostream*& *out*, const TMatrix<*ValueType*>& *m*);

};

Методы:

int operator== (const TMatrix<*ValueType*>& *m*) const;

Назначение: сравнение двух матриц.

Входные данные: матрица, с которой будет производится сравнение.

Выходные данные: результат сравнения (1, если матрицы равны; 0, если матрицы не равны).

int operator!= (const TMatrix<*ValueType*>& *m*) const;

Назначение: сравнение двух матриц.

Входные данные: матрица, с которой будет производится сравнение.

Выходные данные: результат сравнения (0, если матрицы равны; 1, если матрицы не равны).

const *TMatrix*& operator= (const TMatrix<*ValueType*>& *m*);

Назначение: копирование переданной матрицы в исходную.

Входные данные: копируемая матрица.

Выходные данные: матрица – копия переданной матрицы.

*TMatrix* operator+ (const TMatrix<*ValueType*>& *m*);

Назначение: Сложение двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат сложения исходной матрицы и переданной.

*TMatrix* operator- (const TMatrix<*ValueType*>& *m*);

Назначение: вычитание двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат вычитания исходной матрицы и переданной.

*TMatrix* operator\* (const TMatrix<*ValueType*>& *m*);

Назначение: умножение двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат умножения исходной матрицы и переданной.

friend std::*istream*& operator>> (std::*istream*& *in*, TMatrix<*ValueType*>& *m*);

Назначение: ввод данных матрицы (размер и элементов).

Входные данные: поток ввода и матрица.

Выходные данные: поток ввода.

friend std::*ostream*& operator<< (std::*ostream*& *out*, const TMatrix<*ValueType*>& *m*);

Назначение: вывод матрицы на экран.

Входные данные: поток вывода и матрица.

Выходные данные: поток вывода.

# Заключение

В результате данной лабораторной работы были получены практические навыки применения векторов в верхнетреугольных матрицах. На основе полученных знаний в этой предметной области была разработана программа, которая реализует хранение векторов и верхнетреугольных матриц, а также операции над ними. Была произведена проверка корректной работы реализованного функционала с различными наборами данных. В целом, лабораторная работа помогла понять основные принципы работы с верхнетреугольными матрицами и векторами.

# Литературы

1. [Объяснение верхней треугольной и нижней треугольной матрицы (с примерами Python) (skine.ru)](https://skine.ru/articles/554071/)
2. [slide-12.jpg (1024×767) (ppt-online.org)](https://cf.ppt-online.org/files1/slide/b/ByoTdCmb7FGOEwfzxtsWAMprKcgQ62kh3RVZj0S5i/slide-12.jpg)

# Приложения

## Приложение А. Реализация класса TVector

template <typename ValueType>

TVector<ValueType>::TVector(int size, int startIndex) {

  if (size < 0)

    throw std::exception("negative size");

  if (startIndex < 0)

    throw std::exception("negative start index");

  this->size = size;

  this->startIndex = startIndex;

  pVector = new ValueType[size];

  for (int i = 0; i < size; i++)

    pVector[i] = 0;

}

template <typename ValueType>

TVector<ValueType>::TVector(const TVector<ValueType>& v) {

  size = v.size;

  startIndex = v.startIndex;

  pVector = new ValueType[size];

  for (int i = 0; i < size; i++) {

    pVector[i] = v.pVector[i];

  }

}

template <typename ValueType>

TVector<ValueType>::~TVector() {

  delete[] pVector;

}

template <typename ValueType>

ValueType& TVector<ValueType>::operator[](const int i) {

  if (i < 0 || i >= size)

    throw std::exception("out of range");

  return pVector[i];

}

template <typename ValueType>

int TVector<ValueType>::GetSize() const {

  return size;

}

template <typename ValueType>

int TVector<ValueType>::GetStartIndex() const {

  return startIndex;

}

template <typename ValueType>

int TVector<ValueType>::operator== (const TVector<ValueType>& v) const {

  if (size != v.size || startIndex != v.startIndex)

    return 0;

  for (int i = 0; i < size; i++)

    if (pVector[i] != v.pVector[i])

      return 0;

  return 1;

}

template <typename ValueType>

int TVector<ValueType>::operator!= (const TVector<ValueType>& v) const {

  return !(\*this == v);

}

template <typename ValueType>

const TVector<ValueType>& TVector<ValueType>::operator= (const TVector<ValueType>& v) {

  if (this == &v)

    return \*this;

  if (size != v.size) {

    size = v.size;

    delete[] pVector;

    pVector = new ValueType[size];

  }

  startIndex = v.startIndex;

  for (int i = 0; i < size; i++)

    pVector[i] = v.pVector[i];

  return \*this;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator+ (const ValueType val) {

  TVector<ValueType> result(\*this);

  for (int i = 0; i < size; i++)

    result[i] += val;

  return result;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator- (const ValueType val) {

  TVector<ValueType> result(\*this);

  for (int i = 0; i < size; i++)

    result[i] -= val;

  return result;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator\* (const ValueType val) {

  TVector<ValueType> result(\*this);

  for (int i = 0; i < size; i++)

    result[i] \*= val;

  return result;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator+ (const TVector<ValueType>& v) {

  if (size != v.size || startIndex != v.startIndex)

    throw std::exception("diff. sizes");

  TVector<ValueType> result(\*this);

  for (int i = 0; i < size; i++)

    result.pVector[i] = result.pVector[i] + v.pVector[i];

  return result;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator- (const TVector<ValueType>& v) {

  if (size != v.size || startIndex != v.startIndex)

    throw std::exception("diff. sizes");

  TVector<ValueType> result(\*this);

  for (int i = 0; i < size; i++)

    result.pVector[i] = result.pVector[i] - v.pVector[i];

  return result;

}

template <typename ValueType>

double TVector<ValueType>::operator\* (const TVector<ValueType>& v) {

  if (size != v.size || startIndex != v.startIndex)

    throw std::exception("diff. sizes");

  double sum = 0.0;

  for (int i = 0; i < size; i++)

    sum += pVector[i] \* v.pVector[i];

  return sqrt(sum);

}

## Приложение Б. Реализация класса TMatrix

template <typename ValueType>

TMatrix<ValueType>::TMatrix(int n) : TVector<TVector<ValueType>>(n) {

  for (int i = 0; i < n; i++)

    pVector[i] = TVector<ValueType>(n - i, i);

}

template <typename ValueType>

TMatrix<ValueType>::TMatrix(const TMatrix<ValueType>& m) : TVector<TVector<ValueType>>(m) {}

template <typename ValueType>

TMatrix<ValueType>::TMatrix(const TVector<TVector<ValueType>>& v) : TVector<TVector<ValueType>>(v) {}

template <typename ValueType>

int TMatrix<ValueType>::operator== (const TMatrix<ValueType>& m) const {

  return TVector<TVector<ValueType>>::operator==(m);

}

template <typename ValueType>

int TMatrix<ValueType>::operator!= (const TMatrix<ValueType>& m) const {

  return !(\*this == m);

}

template <typename ValueType>

const TMatrix<ValueType>& TMatrix<ValueType>::operator= (const TMatrix<ValueType>& m) {

  return TVector<TVector<ValueType>>::operator=(m);

}

template <typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator+ (const TMatrix<ValueType>& m) {

  return TVector<TVector<ValueType>>::operator+(m);

}

template <typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator- (const TMatrix<ValueType>& m) {

  return TVector<TVector<ValueType>>::operator-(m);

}

template <typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator\* (const TMatrix<ValueType>& m) {

  if (size != m.size)

    throw std::exception("diff. sizes");

  TMatrix<ValueType> res(size);

  for (int i = 0; i < size; i++) {

    for (int j = i; j < size; j++) {

      res[i][j - res.pVector[i].GetStartIndex()] = 0;

      for (int k = i; k <= j; k++) {

        res.pVector[i][j - res.pVector[i].GetStartIndex()] += this->pVector[i][k - this->pVector[i].GetStartIndex()] \* m.pVector[k][j - m.pVector[k].GetStartIndex()];

      }

    }

  }

  return res;

}

## Приложение В. Использование функционала векторов

#include "Vector.h"

#include <iostream>

int main() {

  int size\_1, size\_2, sInd\_1, sInd\_2;

  try {

    std::cout << "Initializing the first vector - a:\n";

    std::cout << " -- enter the size of the first vector:\n    ";

    std::cin >> size\_1;

    std::cout << " -- enter the start index of the first vector:\n    ";

    std::cin >> sInd\_1;

    std::cout << " -- enter the coordinates of your vector:\n";

    TVector<int> a(size\_1, sInd\_1);

    std::cin >> a;

    std::cout << "\nYour vector:\n   ";

    std::cout << a;

    std::cout << "Initializing the second vector - b:\n";

    std::cout << " -- enter the size of the second vector:\n    ";

    std::cin >> size\_2;

    std::cout << " -- enter the start index of the second vector:\n    ";

    std::cin >> sInd\_2;

    std::cout << " -- enter the coordinates of your vector:\n";

    TVector<int> b(size\_2, sInd\_2);

    std::cin >> b;

    std::cout << "\nYour vector:\n   ";

    std::cout << b;

    // сhecking vector operations

    std::cout << "a + b :\n" << a + b << std::endl;

    std::cout << "a - b :\n" << a - b << std::endl;

    std::cout << "a \* b :\n" << a \* b << std::endl;

    // сhecking vector scalar operations

    int value;

    std::cout << "\nEnter some integer value - V:\n   ";

    std::cin >> value;

    std::cout << "a + V :\n" << a + value << std::endl;

    std::cout << "a - V :\n" << a - value << std::endl;

    std::cout << "a \* V :\n" << a \* value << std::endl;

    // checking the assignment and comparison operation

    std::cout << "a == b ?  " << (a == b) << std::endl;

    std::cout << "a != b ?  " << (a != b) << std::endl;

    std::cout << "a = b\n";

    a = b;

    std::cout << "a:\n" << a << std::endl;

    std::cout << "b:\n" << b << std::endl;

    std::cout << "a == b ?  " << (a == b) << std::endl;

    std::cout << "a != b ?  " << (a != b) << std::endl;

  }

  catch (const std::*exception*& exp) {

    std::cerr << exp.what() << std::endl;

  }

  return 0;

}

## Приложение Г. Использование функционала верхнетреугольных матриц

#include "Vector.h"

#include "Matrix.h"

#include <iostream>

int main() {

  TMatrix<int> m(5);

  TMatrix<int> d(5);

  // сhecking matrix operations

  try {

    std::cin >> m;

    std::cin >> d;

    std::cout << "Your matrixes:\n";

    std::cout << "m:\n" << m << std::endl;

    std::cout << "d:\n" << d << std::endl;

    std::cout << "\nCompare the results of all operations with your matrices:\n";

    std::cout << "m \* d :\n" << m \* d << std::endl;

    std::cout << "m + d :\n" << m + d << std::endl;

    std::cout << "m - d :\n" << m - d << std::endl;

  }

  catch (const std::*exception*& exp) {

    std::cerr << exp.what() << std::endl << std::endl;

  }

  // checking the assignment and comparison operation

  try {

    std::cout << "m == d ?  " << (m == d) << std::endl;

    std::cout << "m != d ?  " << (m != d) << std::endl;

    std::cout << "m = d\n";

    m = d;

    std::cout << "m:\n" << m << std::endl;

    std::cout << "d:\n" << d << std::endl;

    std::cout << "m == d ?  " << (m == d) << std::endl;

    std::cout << "m != d ?  " << (m != d) << std::endl;

  }

  catch (const std::*exception*& exp) {

    std::cerr << exp.what() << std::endl;

  }

  return 0;

}