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 механики**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Структура хранения матриц специального вида»**

**Выполнил(а):** студент(ка)

Группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фролова Е.А./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc154100637)

[1 Постановка задачи 4](#_Toc154100638)

[2 Руководство пользователя 5](#_Toc154100639)

[2.1 Приложение для демонстрации работы класса вектор 5](#_Toc154100640)

[2.2 Приложение для демонстрации работы класса верхнетреугольных матриц 6](#_Toc154100641)

[3 Руководство программиста 7](#_Toc154100642)

[3.1 Описание алгоритмов 7](#_Toc154100643)

[3.1.1 Векторы 7](#_Toc154100644)

[3.1.2 Верхнетреугольные матрицы 9](#_Toc154100645)

[3.2 Описание программной реализации 13](#_Toc154100646)

[3.2.1 Описание класса TVector 13](#_Toc154100647)

[3.2.2 Описание класса TMatrix 15](#_Toc154100648)

[Заключение 17](#_Toc154100649)

[Литература 18](#_Toc154100650)

[Приложения 19](#_Toc154100651)

[Приложение А. Реализация класса TVector 19](#_Toc154100652)

[Приложение Б. Реализация класса TMatrix 21](#_Toc154100653)

# Введение

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

Цель – изучение одного из возможных подходов к хранению и обработке матриц.

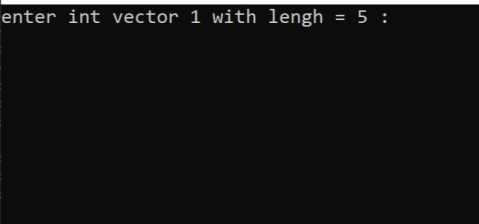
Задача: создания программных средств, поддерживающих эффективное хранение матриц специального вида (верхнетреугольных) и выполнение основных операций над ними:

* сложение/вычитание;
* умножение;
* копирование;
* сравнение.

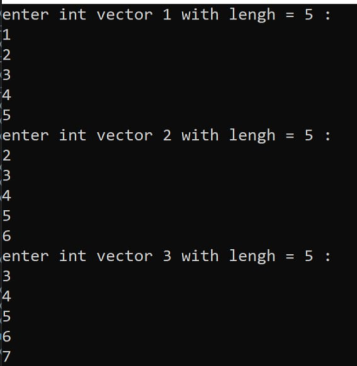
# Руководство пользователя

## Приложение для демонстрации работы класса вектор

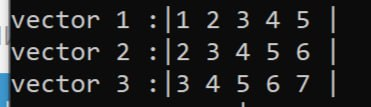
1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (рис. 1).



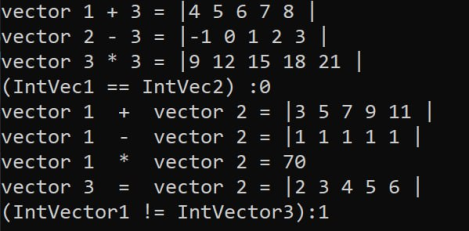
1. Основное окно прграммы.
2. Введём элементы первого вектора.
3. Такую же операцию выполняем для двух последующих векторов (рис. 2).



1. Основное окно программы.
2. Далее программа выводит введённые пользователем векторы (рис. 3).



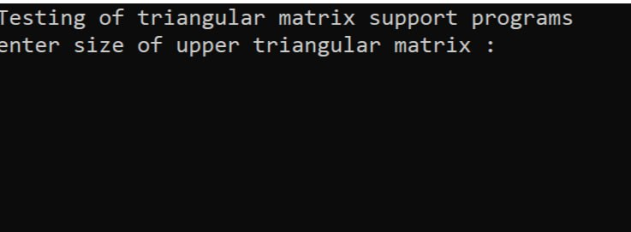
1. Результат ввода векторов.
2. Также программа выводит результат операций, проведённых с вектором такие как: скалярные операции (прибавление константы, вычитание константы, умножение на константу) (рис. 4), векторные операции (сложение векторов, разность векторов, скалярное произведение векторов) (рис. 4), проверка векторов на равенство (неравенство) (рис. 4).



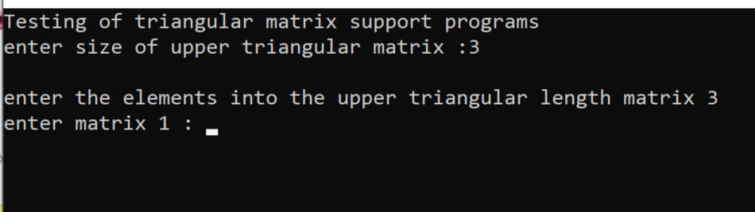
1. Вывод результата операций с векторами.
2. Работа программы окончена.

## Приложение для демонстрации работы класса верхнетреугольных матриц

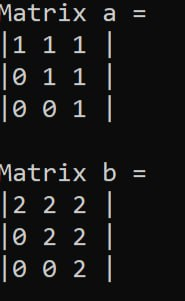
1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (рис. 5).



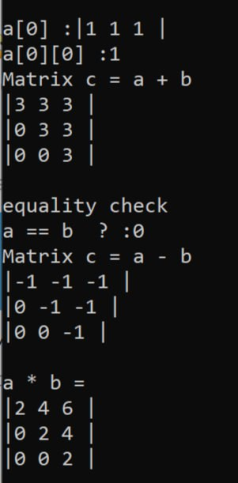
1. Основное окно программы.
2. Введём размер матрицы (рис. 6). Далее нам требуется ввести элементы матрицы.



1. Запрос на ввод элементов матрицы.
2. Выполним ту же самую операцию для второй матрицы. В качестве результата получим две ранее введённые матрицы (рис. 7).



1. Введенные матрицы.
2. Также программа выведет результаты произведенных операций с матрицами (сложение матриц, разность, произведение) (рис. 8).



1. Операции над матрицами.
2. Работа программы окончена.

# Руководство программиста

## Описание алгоритмов

### Векторы

Вектор – в математике – набор (чисел, математических выражений), состоящий из n элементов (должна быть возможность хранить элементы в векторе любого типа. Как целочисленные, так и с плавающей запятой).

Для структуры данных Вектор предлагается реализовать следующие операции:

* сравнение;
* прибавление/вычитание скаляра;
* умножение на скаляр;
* сложение/вычитание векторов;
* скалярное произведение векторов;

Принцип работы:

**Сравнение**

1. Входные данные: два вектора A и B.

2. Сравнение векторов: для сравнения двух векторов A и B, мы сравниваем каждый элемент вектора A с соответствующим элементом вектора B. Если все элементы равны, то векторы считаются равными.

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V1 | 1 | 2 | 3 | 4 | 5 |
| V2 | 1 | 2 | 3 | 3 | 5 |

Векторы не равны, в результате будет выведен ноль.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V1 | 1 | 2 | 3 | 4 | 5 |
| V2 | 1 | 2 | 3 | 4 | 5 |

Векторы равны, в результате будет выведена единица.

**Прибавление/вычитание скаляра.**

1. Входные данные: вектор A и скаляр k.
2. Прибавление(вычитание) скаляра к вектору: для прибавления скаляра k к вектору A, мы просто прибавляем значение k к каждому элементу вектора A (для вычитания скаляра k из вектора A, мы просто вычитаем значение k из каждого элемента вектора A).

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 1 | 3 | 5 | 7 | 9 |

Прибавим к вектору скаляр: 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 6 | 8 | 10 | 12 | 14 |

В результате каждый элемент вектора увеличился на скаляр.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 1 | 3 | 5 | 7 | 9 |

Вычтем из вектора скаляр: 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | -4 | -2 | 0 | 2 | 4 |

В результате каждый элемент вектора уменьшился на скаляр.

**Умножение на скаляр.**

1. Входные данные: вектор A и скаляр k.
2. Умножение вектора на скаляр: для умножение вектора А на скаляр k, мы просто умножаем каждый элементу вектора A на значение k.

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 1 | 2 | 3 | 4 | 5 |

Умножим вектор на скаляр: 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 5 | 10 | 15 | 20 | 25 |

В результате каждый элемент вектора увеличился в пять раз.

**Сложение/вычитание векторов.**

1. Входные данные: вектор A и вектор B.
2. Сложение векторов: для сложения векторов A и B, мы просто складываем соответствующие элементы каждого вектора.
3. Вычитание векторов: для вычитания векторов A и B, мы просто вычитаем соответствующие элементы каждого вектора.

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V1 | 1 | 2 | 3 | 4 | 5 |
| V2 | 1 | 1 | 1 | 3 | 1 |

Результатом сложения двух векторов будет вектор:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 2 | 3 | 4 | 7 | 6 |

Результатом разности двух векторов (из первого вычитаем второй) будет вектор:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vec | 0 | 1 | 2 | 1 | 4 |

**Скалярное произведение векторов.**

1. Входные данные: вектор A и вектор B.
2. Скалярное произведение векторов: для вычисления скалярного произведения векторов A и B, мы умножаем соответствующие элементы каждого вектора и складываем результаты.

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V1 | 1 | 2 | 3 | 4 | 5 |
| V2 | 2 | 3 | 4 | 5 | 6 |

Скалярное произведение векторов в таком случае будет рассчитываться:

V1 \* v2 = 1\*2 + 2\*3 + 3\*4 + 4\*5 + 5\*6 = 70

### Верхнетреугольные матрицы

Матрица – в математике – прямоугольная таблица каких-либо элементов (чисел, математических выражений), состоящая из m строк и n столбцов. Над матрицей можно производить действия по правилам матричной алгебры. При m=n матрица называется квадратной, а число n – её порядком. Набор элементов матрицы (a11, a22, …, ann) называется главной диагональю.

Верхнетреугольной называется матрица, в которой все элементы под главной диагональю равны нулю.

Определение матрицы возможно также через понятие Вектор. Вектор – в математике – набор ai (чисел, математических выражений), состоящий из n элементов. Тогда Матрица из m строк и n столбцов может быть определена как Вектор из n элементов, где каждый элемент, в свою очередь, является вектором из m элементов.

Для верхнетреугольной матрицы имеет смысл задать структуру данных таким образом, чтобы исключить хранение нулевых элементов. Определение матрицы через вектор позволяет сделать это наилучшим образом.

Очевидно сходство в задании структуры данных Вектор, как набора элементов, связанных отношением следования, и структуры данных Матрица, как набора элементоввекторов, связанных отношением следования. Этот факт позволяет единообразно организовать алгоритмы обработки векторов и матриц, а, следовательно, использовать при разработке требуемых классов механизм наследования.

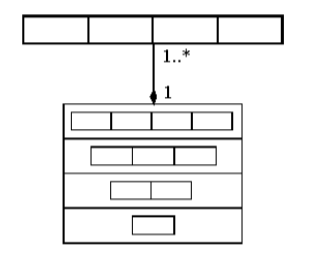


Рис. 9. Матрица как вектор векторных элементов.

Для структуры данных Матрица предлагается реализовать следующие операции:

* сравнение.
* сложение/вычитание матриц.
* умножение матриц.

Принцип работы:

**Сравнение матриц**

Алгоритм сравнения двух верхнетреугольных матриц, представленных в виде вектора векторов:

* 1. Начнем сравнение с первой строки (первого вектора) и будем постепенно двигаться к последней строке (последнему вектору).
  2. Для каждой строки (вектора) сравниваемых матриц:
  3. Сравниваем соответствующие элементы каждой строки. Так как в каждой следующей строке на один элемент меньше, то количество элементов для сравнения в каждом векторе должно быть равно (размер матрицы – i)), где i - номер строки (нумерация с нуля). Если хотя бы один элемент не совпадает, то матрицы не равны.
  4. Если все элементы в каждой строке совпадают, то матрицы – равны.

Пример:

Первая матрица:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
|  | 4 | 5 |
|  |  | 6 |

Вторая матрица:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
|  | 4 | 5 |
|  |  | 6 |

Результатом будет: 1. Обе матрицы равны.

**Сложение/вычитание матриц.**

Конечно, вот подробный алгоритм сложения и вычитания для верхнетреугольных матриц, представленных в виде вектора векторов:

1. Проверить, что обе матрицы имеют одинаковый размер (т.е. одинаковое количество векторов в векторе векторов). Если размеры матриц различны, операции сложения и вычитания невозможны.
2. Создать новую пустую матрицу того же размера, что и исходные матрицы, для хранения результата операции.
3. Для каждого вектора i (нумерация с 0) в матрицах A и B выполнить следующее:

* Определить количество элементов, участвующих в операции, равное (размер матрицы - i) (нумерация с нуля).
* Для операции сложения: сложить соответствующие элементы A[i][j] и B[i][j] (элемент принадлежит i-му вектору вектора матрицы и находится по адресу j в i-м векторе) и поместить результат в соответствующий элемент новой матрицы.
* Для операции вычитания: вычесть соответствующие элементы B[i][j] из A[i][j] и поместить результат в соответствующий элемент новой матрицы.

1. Вернуть новую матрицу как результат операции сложения или вычитания.

Пример:

Первая матрица:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
|  | 2 | 3 |
|  |  | 4 |

Вторая матрица:

|  |  |  |
| --- | --- | --- |
| 4 | 2 | 7 |
|  | 1 | 3 |
|  |  | 3 |

Результатом сложения двух матриц будет матрица:

|  |  |  |
| --- | --- | --- |
| 5 | 4 | 10 |
|  | 3 | 6 |
|  |  | 7 |

Результатом разности двух матриц (из первой вычитаем вторую) будет матрица:

|  |  |  |
| --- | --- | --- |
| -3 | 0 | -4 |
|  | 1 | 0 |
|  |  | 1 |

**Умножение матриц**

Алгоритм умножения матриц, представленных в виде вектора векторов:

* 1. Проверить, что количество столбцов в первой матрице (число элементов в первом векторе) равно количеству строк во второй матрице (число векторов). Если это условие не выполняется, операция умножения невозможна (в условиях нашей задачи матрица квадратная).
  2. Создать новый пустой вектор векторов для хранения результата умножения. Количество строк новой матрицы равно количеству строк первой матрицы, а количество столбцов равно количеству столбцов второй матрицы.
  3. Для каждой строки i (нумерация с 0) в первой матрице выполнить следующее:
* Определить количество элементов, участвующих в операции, равное (размер матрицы - i).
* Для каждого столбца j (нумерация с 1) во второй матрице выполнить следующее:
* Вычислить скалярное произведение i-й строки первой матрицы и j-го столбца второй матрицы.
* Поместить результат скалярного произведения в соответствующий элемент новой матрицы.
  1. Вернуть новую матрицу как результат умножения.

Пример:

Первая матрица:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
|  | 2 | 3 |
|  |  | 4 |

Вторая матрица:

|  |  |  |
| --- | --- | --- |
| 4 | 2 | 7 |
|  | 1 | 3 |
|  |  | 3 |

Результатом умножения двух матриц будет матрица:

|  |  |  |
| --- | --- | --- |
| 4 | 4 | 22 |
|  | 2 | 15 |
|  |  | 12 |

## Описание программной реализации

### Описание класса TVector

**template <class T>**

**class TVector**

**{**

**protected:**

**T\* pVector;**

**int Size;**

**int StartIndex;**

**public:**

**TVector(int Size = 10, int StartIndex = 0);**

**TVector(const TVector<T>& v);**

**virtual ~TVector<T>();**

**int GetSize() { return Size; }**

**int GetStartIndex() { return StartIndex; }**

**T& operator[](int pos);**

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

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

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

**TVector<T> operator+(const T& val);**

**TVector<T> operator-(const T& val);**

**TVector<T> operator\*(const T& val);**

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

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

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

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

**friend ostream& operator<<(ostream& out, const TVector<T>& v);**

**};**

Назначение: представление вектора.

Поля:

**Size** – размер вектора.

**StartIndex** – индекс первого элемента вектора.

**pVector** – память для представления вектора.

Методы:

**int GetSize();**

Назначение: получение получить длину вектора.

Входные параметры: нет.

Выходные параметры: размер вектора.

**int GetStartIndex();**

Назначение: получение индекса первого элемента вектора.

Входные параметры: нет.

Выходные параметры: индекс первого элемента вектора.

**T& operator[](int pos);**

Назначение: получение элемента вектора по индексу.

Входные параметры: **pos**-индекс элемента.

Выходные параметры: элемента вектора по заданному индексу.

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

Назначение: проверка векторов на равенство.

Входные параметры: ссылка на элемент типа **TVector**.

Выходные параметры: результат сравнения двух векторов.

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

Назначение: проверка векторов на неравенство.

Входные параметры: ссылка на элемент типа **TVector**.

Выходные параметры: результат сравнения двух векторов.

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

Назначение: присваивание одним элементом типа **TVector** значениий полей другого элемента типа **TVector**.

Входные параметры: константная ссылка на элемент типа **TVector**.

Выходные параметры: элемент типа **TVector** - результат присваивания первому

элементу типа **TVector** значения второго элемента **TVector**.

**TVector<T> operator+(const T& val);**

Назначение: прибавление к вектору константы.

Входные параметры: число.

Выходные параметры: элемент типа **TVector** - результат сложения элемента типа **TVector** с константой.

**TVector<T> operator-(const T& val);**

Назначение: разность между вектором и константой.

Входные параметры: число.

Выходные параметры: элемент типа **TVector** - результат разности между элементом типа **TVector** и константой.

**TVector<T> operator\*(const T& val);**

Назначение: произведение между вектором и константой.

Входные параметры: число.

Выходные параметры: элемент типа **TVector** - результат произведения элемента типа **TVector** с константой.

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

Назначение: сложение двух векторов.

Входные параметры: вектор.

Выходные параметры: элемент типа **TVector** - результат сложения двух

элементов типа **TVector**.

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

Назначение: разность двух векторов.

Входные параметры: вектор.

Выходные параметры: элемент типа **TVector** - результат разности двух

элементов типа **TVector**.

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

Назначение: скалярное произведение двух векторов.

Входные параметры: вектор.

Выходные параметры: элемент типа **T** - результат скалярного произведения двух

элементов типа **TVector**.

### Описание класса TMatrix

template <class T>

class TMatrix : public TVector<TVector<T>> {

public:

TMatrix<T>(int s = 10);

TMatrix<T>(const TMatrix<T>& mt);

TMatrix<T>(const TVector<TVector<T> >& mt);

TVector<T>& operator[](const int index);

int operator==(const TMatrix<T>& mt) const;

int operator!=(const TMatrix<T>& mt) const;

TMatrix<T>& operator= (const TMatrix<T>& mt);

TMatrix<T> operator+ (const TMatrix<T>& mt);

TMatrix<T> operator- (const TMatrix<T>& mt);

TMatrix<T> operator\* (const TMatrix<T>& mt);

friend istream& operator>>(istream& in, TMatrix<T>& mt);

friend ostream& operator<<(ostream& out, const TMatrix<T>& mt);

};

Назначение: представление верхнетреугольной матрицы.

Методы:

TVector<T>& operator[](const int index);

Назначение: получение элемента матрицы по индексу.

Входные параметры: **index**-индекс элемента.

Выходные параметры: элемента матрицы по заданному индексу.

int operator==(const TMatrix<T>& mt) const;

Назначение: проверка матриц на равенство.

Входные параметры: ссылка на элемент типа **TMatrix**.

Выходные параметры: результат сравнения двух матриц.

int operator!=(const TMatrix<T>& mt) const;

Назначение: проверка матриц на неравенство.

Входные параметры: ссылка на элемент типа **TMatrix**.

Выходные параметры: результат сравнения двух матриц.

TMatrix<T>& operator= (const TMatrix<T>& mt);

Назначение: присваивание одним элементом типа **TMatrix** значениий полей другого элемента типа **TMatrix**.

Входные параметры: константная ссылка на элемент типа **TMatrix**.

Выходные параметры: элемент типа **TMatrix** - результат присваивания первому

элементу типа **TMatrix** значения второго элемента типа **TMatrix**.

TMatrix<T> operator+ (const TMatrix<T>& mt);

Назначение: сложение двух матриц.

Входные параметры: матрица.

Выходные параметры: элемент типа **TMatrix** - результат сложения двух

элементов типа **TMatrix**.

TMatrix<T> operator- (const TMatrix<T>& mt);

Назначение: разность между двумя матрицами.

Входные параметры: матрица.

Выходные параметры: элемент типа **TMatrix** - результат разности меду двумя

элементами типа **TMatrix**.

TMatrix<T> operator\* (const TMatrix<T>& mt);

Назначение: произведение двух матриц.

Входные параметры: матрица.

Выходные параметры: элемент типа **TMatrix** - результат произведения двух

элементов типа **TMatrix**.

# Заключение

Мы изучили один из возможных подходов к хранению и обработке матриц. Создали программных средства, поддерживающие эффективное хранение матриц специального вида (верхнетреугольных) и выполнение основных операций над ними:

* сложение/вычитание;
* умножение;
* копирование;
* сравнение;

# Литература

1. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# Приложения

## Приложение А. Реализация класса TVector

template <class T>

TVector <T> ::TVector(int Size, int StartIndex)

{

if (Size <= 0)

throw"Error";

if (StartIndex < 0)

throw"Error";

this->StartIndex = StartIndex;

this->Size = Size;

pVector = new T[Size];

}

template <class T>

TVector <T> ::TVector(const TVector<T>& v)

{

Size = v.Size;

StartIndex = v.StartIndex;

pVector = new T[Size];

for (int i = 0; i < Size; i++)

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

}

template <class T>

TVector <T>::~TVector()

{

if (pVector != NULL)

{

delete[] pVector;

pVector = NULL;

Size = 0;

}

}

template <class T>

T& TVector <T>:: operator[](int pos)

{

if (pos < StartIndex || pos >= Size + StartIndex)

throw"Error";

return pVector[pos - StartIndex];

}

template<class T>

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

{

if (StartIndex != v.StartIndex)

return 0;

if (Size != v.Size)

return 0;

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

if (pVector[i] != v.pVector[i])

return 0;

return 1;

}

template<class T>

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

{

return !(\*this == v);

}

template<class T>

const TVector<T>& TVector<T>::operator=(const TVector<T>& v)

{

if (this != &v)

{

if (Size != v.Size)

{

delete[] pVector;

Size = v.Size;

pVector = new T[v.Size];

}

StartIndex = v.StartIndex;

for (int i = 0; i < Size; i++)

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

}

return \*this;

}

template<class T>

TVector<T> TVector<T>:: operator+ (const T& val)

{

TVector<T> tmp(\*this);

for (int i = 0; i < Size; i++)

{

tmp.pVector[i] += val;

}

return tmp;

}

template<class T>

TVector<T> TVector<T>:: operator- (const T& val)

{

TVector<T> tmp(\*this);

for (int i = 0; i < Size; i++)

{

tmp.pVector[i] -= val;

}

return tmp;

}

template<class T>

TVector<T> TVector<T>:: operator\* (const T& val)

{

TVector<T> tmp(\*this);

for (int i = 0; i < Size; i++)

{

tmp.pVector[i] \*= val;

}

return tmp;

}

template <class T>

TVector<T> TVector<T>:: operator+(const TVector<T>& v)

{

if (Size != v.Size || StartIndex != v.StartIndex)

throw"Error";

TVector<T> tmp(\*this);

for (int i = 0; i < Size; i++)

{

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

}

return tmp;

}

template <class T>

TVector<T> TVector<T>:: operator-(const TVector<T>& v)

{

if (Size != v.Size || StartIndex != v.StartIndex)

throw"Error";

TVector<T> tmp(\*this);

for (int i = 0; i < Size; i++)

{

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

}

return tmp;

}

template <class T>

T TVector <T>:: operator\* (const TVector<T>& v)

{

if (Size != v.Size || StartIndex != v.StartIndex)

throw "Error";

T res = T();

for (int i = 0; i < Size; i++)

{

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

}

return res;

}

## Приложение Б. Реализация класса TMatrix

template <class T>

TMatrix<T>::TMatrix(const TMatrix<T>& mt) :TVector<TVector<T>>((TVector<TVector<T>>)mt) {}

template <class T>

TMatrix<T>::TMatrix<T>(const TVector<TVector<T> >& mt) : TVector<TVector<T>>(mt) {}

template <class T>

int TMatrix<T>:: operator==(const TMatrix<T>& mt) const

{

return TVector<TVector<T>>::operator==(mt);

}

template <class T>

int TMatrix<T>:: operator!=(const TMatrix<T>& mt) const

{

return !(\*this == mt);

}

template <class T>

TVector<T>& TMatrix<T>:: operator[](const int index)

{

return TVector<TVector<T>>::operator[](index);

}

template <class T>

TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& mt)

{

if (this != &mt)

{

if (this->Size != mt.Size)

{

delete[] this->pVector;

this->pVector = new TVector<T>[mt.Size];

}

this->Size = mt.Size;

this->StartIndex = mt.StartIndex;

for (int i = 0; i < this->Size; i++)

this->pVector[i] = mt.pVector[i];

}

return \*this;

}

template <class T>

TMatrix<T> TMatrix<T>::operator+ (const TMatrix<T>& mt)

{

return TVector<TVector<T>>::operator+(mt);

}

template <class T>

TMatrix<T> TMatrix<T>::operator- (const TMatrix<T>& mt)

{

return TVector<TVector<T>>::operator-(mt);

}

template <class T>

TMatrix<T> TMatrix<T>:: operator\* (const TMatrix<T>& mt)

{

if ((this->Size != mt.Size)||(this->StartIndex!=mt.StartIndex))

throw"Error";

TMatrix<T> tmp(this->Size);

for (int i = 0; i < this->Size; i++)

for (int j = tmp[i].GetStartIndex(); j < this->Size; j++)

tmp[i][j] = T(NULL);

for (int i = 0; i < this->Size; i++)

{

for (int j = tmp[i].GetStartIndex(); j < this->Size; j++)

{

T sum = 0;

for (int k = tmp[i].GetStartIndex(); k <= j; k++) {

sum += (\*this)[i][k] \* mt.pVector[k][j];

}

tmp.pVector[i][j] = sum;

}

}

return tmp;

};