

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
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:

«Векторы и матрицы»

Выполнил(а): студент(ка) группы
3822Б1ФИ2

_____ / Савченко М.П./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП
_____ / Кустикова В.Д./

Подпись

Нижний Новгород
2023

Содержание

Введение.....	3
1 Постановка задачи.....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы векторов	5
2.2 Приложение для демонстрации работы матриц	6
3 Руководство программиста	8
3.1 Описание алгоритмов	8
3.1.1 Векторы	8
3.1.2 Матрицы.....	9
3.2 Описание программной реализации	11
3.2.1 Описание класса TVector.....	11
3.2.2 Описание класса TMatrix.....	14
Заключение	17
Литература	18
Приложения	19
Приложение А. Реализация класса TVector	19
Приложение Б. Реализация класса TMatrix	22

Введение

Векторы и матрицы - это фундаментальные математические концепции, которые имеют широкий спектр применений в различных областях. Вот некоторые из них:

1. Математика и наука:

- Векторы и матрицы используются для решения систем линейных уравнений и анализа линейных преобразований, что имеет важное значение в линейной алгебре.
- Векторы могут представлять геометрические сущности, такие как точки, направления, скорости и ускорения.
- Матрицы используются в линейной алгебре для решения линейных систем и многих других задач.

2. Физика и инженерия:

- Векторы широко применяются для моделирования и анализа движения, сил и моментов в механике.
- Матрицы используются для решения систем дифференциальных уравнений, описывающих физические явления.

3. Компьютерная графика и компьютерное зрение:

- Векторы используются для описания трехмерных объектов и графических преобразований, таких как вращение и масштабирование.
- Матрицы применяются для преобразований координат и трансформаций объектов.

4. Машинное обучение и обработка данных:

- Векторы широко используются для представления данных, таких как признаки и метки классов в машинном обучении.
- Матрицы используются для хранения и обработки данных, включая операции линейной алгебры, используемые в алгоритмах машинного обучения.

5. Экономика и финансы:

- Векторы могут представлять портфели инвестиций, расходы или доходы, что позволяет анализировать финансовые данные.
- Матрицы применяются, например, для оценки рисков и доходности портфелей.

Векторы и матрицы предоставляют мощные инструменты для описания и решения разнообразных задач в науке, технике и многих других областях.

1 Постановка задачи

Цель – реализовать шаблонный класс для представления вектора TVector с параметром начального индекса, и на его основе реализовать шаблонный класс для представления верхнетреугольной матрицы TMatrix.

Задачи при реализации класса TVector:

1. Описать и реализовать конструктор, конструктор копирования, деструктор.
2. Описать и реализовать методы возврата длины вектора и начального индекса.
3. Перегрузить операторы индексации, присваивания, сравнения.
4. Перегрузить операторы сложения, вычитания, умножения вектора со скаляром.
5. Перегрузить операторы сложения, вычитания с вектором.
6. Перегрузить скалярное произведение векторов.
7. Перегрузить операции ввода и вывода.

Задачи при реализации класса TMatrix:

1. Реализовать TMatrix как вектор, элементами которого являются вектора.
2. Предусмотреть то, что матрица верхнетреугольная.
3. Описать и реализовать конструктор, конструктор копирования, конструктор преобразования типа.
4. Перегрузить операторы сложения и вычитания матриц.
5. Перегрузить оператор умножения верхнетреугольных матриц.
6. Перегрузить операции ввода и вывода.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

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

```
D:\UNN\My_Projects\02_lab.TMatrix\mp2-practice\SavchenkoMP\02_lab\sln\bin>sample_tvector.exe
TVector

Enter the values of the elements of two vectors with length 4.
input example:
1 2 3 4 4 3 2 1
_
```

Рис. 1. Основное окно программы sample_tvector.exe

2. Введите значения координат двух четырехмерных векторов, например, «1 2 3 4 3 2 1». После ввода в окне напечатаются новые данные, показанные ниже (рис. 2).

```
TVector

Enter the values of the elements of two vectors with length 4.
input example:
1 2 3 4 4 3 2 1

1 2 3 4 4 3 2 1

v1 = | 1 2 3 4 |
v2 = | 4 3 2 1 |

scalar = 2
v1 * scalar = | 2 4 6 8 |
v1 + scalar = | 3 4 5 6 |
v1 - scalar = | -1 0 1 2 |

v1 + v2 = | 5 5 5 5 |
v1 - v2 = | -3 -1 1 3 |
v1 * v2 = 20

(v1 == v2) = 0
(v1 != v2) = 1

v1 = | 1 2 3 4 |
v1[3] = 4
v1.GetSize() = 4
v1.GetStartIndex() = 0

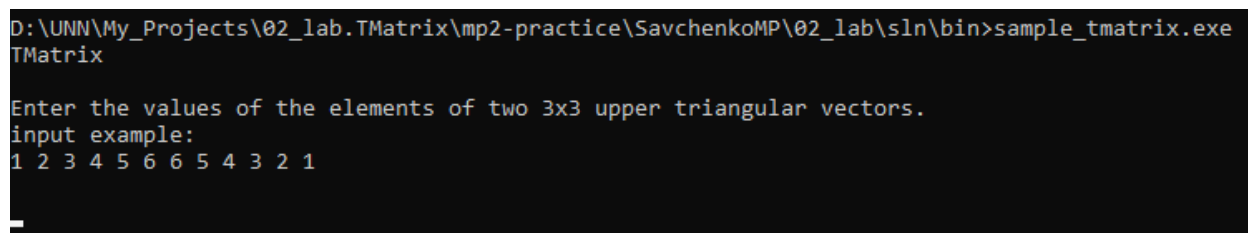
Example of vector with startIndex:
v3 = | 0 0 1 2 3 4 |
v3.GetStartIndex() = 2
v3[2] = 1
```

Рис. 2. Выведенными данные программы sample_tvector.exe

- Здесь вы можете ознакомиться с основными операциями над векторами: сложение, вычитание, умножение со скаляром; сложение, вычитание с вектором; скалярное произведение; пример вектора с ненулевым начальным индексом.

2.2 Приложение для демонстрации работы матриц

- Запустите приложение с названием sample_tmatrix.exe. В результате появится окно, показанное ниже (рис. 3).



```
D:\UNN\My_Projects\02_lab.TMatrix\mp2-practice\SavchenkoMP\02_lab\sln\bin>sample_tmatrix.exe
TMatrix

Enter the values of the elements of two 3x3 upper triangular vectors.
input example:
1 2 3 4 5 6 6 5 4 3 2 1
```

Рис. 3. Основное окно программы sample_tmatrix.exe

- Введите значения координат двух верхнетреугольных матриц 3 на 3, например, «1 2 3 4 5 6 6 5 4 3 2 1». После ввода в окне напечатаются новые данные, показанные ниже (рис. 4).

```

Enter the values of the elements of two 3x3 upper triangular vectors.
input example:
1 2 3 4 5 6 6 5 4 3 2 1
1 2 3 4 5 6 6 5 4 3 2 1

m1 =
| 1 2 3 |
| 0 4 5 |
| 0 0 6 |

m2 =
| 6 5 4 |
| 0 3 2 |
| 0 0 1 |

m1[1] = | 0 4 5 |
m1[1][1] = 4

m1 + m2 =
| 7 7 7 |
| 0 7 7 |
| 0 0 7 |

m1 - m2 =
| -5 -3 -1 |
| 0 1 3 |
| 0 0 5 |

m1 * m2 =
| 6 11 11 |
| 0 12 13 |
| 0 0 6 |

(m1 == m2) = 0
(m1 != m2) = 1

```

Рис. 4. Выведенные данные программы sample_tmatrix.exe

3. Здесь вы можете ознакомиться с основными операциями над верхнетреугольными матрицами: индексация элемента-вектора и индексация элемента-координаты; сложение, вычитание, умножение верхнетреугольных матриц; сравнение верхнетреугольных матриц.

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Векторы

Вектор представляет собой структуру данных, используемую для хранения элементов одного и того же типа. Он описывается как указатель на массив элементов этого типа, с указанием стартового индекса и количества элементов в нем. Если стартовый индекс отличен от нуля, то предполагается, что элементы с индексами меньше стартового содержат нейтральные значения.

Операции, поддерживаемые вектором, включают:

- Векторно-скалярные операции: включают в себя сложение, вычитание и умножение вектора на элемент того же типа данных.
- Векторно-векторные операции: включают в себя сложение и вычитание векторов того же типа.
- Скалярное произведение векторов: это операция, позволяющая вычислить сумму произведений соответствующих элементов двух векторов.
- Сравнение векторов на равенство или неравенство.

Эти операции являются основными для работы с векторами и позволяют выполнять разнообразные вычисления и сравнения в рамках данной структуры данных.

Векторно-скалярные операции

В векторно-скалярных операциях выполняется поэлементное сложение, вычитание и умножение вектора на скаляр – элемент того же типа, который хранится в векторе.

Пример: $v = (1, 2, 3, 4)$

$$v + 2 = (1 + 2, 2 + 2, 3 + 2, 4 + 2) = (3, 4, 5, 6)$$

$$v - 2 = (1 - 2, 2 - 2, 3 - 2, 4 - 2) = (-1, 0, 1, 2)$$

$$v * 2 = (1 * 2, 2 * 2, 3 * 2, 4 * 2) = (2, 4, 6, 8)$$

Векторно-векторные операции

В результирующем векторе каждый элемент получается путем сложения или вычитания элементов входящих векторов с тем же индексом.

Пример: $v_1 = (1, 2, 3, 4), v_2 = (1, 1, 1, 1)$

$$v_1 + v_2 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 + 1 \\ 2 + 1 \\ 3 + 1 \\ 4 + 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

$$v_1 - v_2 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1-1 \\ 2-1 \\ 3-1 \\ 4-1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix}$$

Скалярное произведение

Скалярное произведение векторов, заданных своими координатами, равно сумме произведений соответствующих координат.

$$\vec{a} * \vec{b} = \sum_{i=0}^{n-1} a_i * b_i$$

Пример: $v_1 = (1, 2, 3), v_2 = (2, 2, 2)$

$$v_1 * v_2 = 1 * 2 + 2 * 2 + 3 * 2 = 12$$

Сравнение векторов

Вектора сравниваются поэлементно: они считаются равными, если каждый элемент одного вектора равен элементу второго вектора с тем же индексом, иначе они считаются неравными.

3.1.2 Матрицы

Матрица - это структура данных, которая представляет собой набор векторов, где каждый вектор содержит элементы одного типа. Верхнетреугольная матрица - это матрица, у которой все элементы, находящиеся ниже главной диагонали, равны нулю или какому-то нейтральному элементу.

Операции, которые поддерживаются для матриц, включают в себя:

- Сложение и вычитание матриц одного типа: элементы матриц складываются или вычитаются поэлементно, если матрицы имеют одинаковый тип данных.
- Произведение матриц одного типа: это операция, которая позволяет умножать матрицы друг на друга в соответствии с определенными правилами умножения матриц.
- Сравнение матриц на равенство или неравенство: матрицы могут быть сравниваемыми поэлементно, чтобы определить, равны они или нет.

Эти операции позволяют выполнять различные вычисления и сравнения с матрицами, что полезно во многих областях, включая линейную алгебру и численное моделирование.

Сложение и вычитание матриц

Результирующая матрица формируется путем поэлементного сложения или вычитания соответствующих элементов входящих матриц с одинаковыми индексами.

$$C_{n \times m} = A_{n \times m} \pm B_{n \times m}$$
$$c_{ij} = a_{ij} \pm b_{ij} \quad (i = \overline{0, n-1}, j = \overline{0, m-1})$$

Пример:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$
$$C = A + B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & 0 & 7 \end{pmatrix}$$
$$D = A - B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix} - \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 3 & 4 \\ 0 & 0 & 5 \end{pmatrix}$$

Умножение матриц

Результатом умножения матриц $A_{m \times n}$ и $B_{n \times k}$ будет матрица $C_{m \times k}$ такая, что элемент матрицы C , стоящий в i -той строке и j -том столбце (c_{ij}), равен сумме произведений элементов i -той строки матрицы A на соответствующие элементы j -того столбца матрицы B :

$$c_{ij} = \sum_{t=0}^{n-1} a_{it} * b_{tj}, \quad (i = \overline{0, m-1}, j = \overline{0, k-1})$$

Пример:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}, B = \begin{pmatrix} 6 & 5 & 4 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$
$$C = A * B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix} * \begin{pmatrix} 6 & 5 & 4 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 6 & 11 & 11 \\ 0 & 12 & 13 \\ 0 & 0 & 6 \end{pmatrix}$$

Сравнение матриц

Матрицы сравниваются поэлементно: они считаются равными, если каждый элемент одной матрицы равен элементу второй матрицы с тем же индексом, иначе они считаются неравными.

3.2 Описание программной реализации

3.2.1 Описание класса TVector

```
template <typename T>
class TVector {
protected:
    int size;
    int startIndex;
    T* pMem;
public:
    TVector(int _size = 10, int _startIndex = 0);
    TVector(const TVector<T>& v);
    virtual ~TVector();

    int GetSize() const noexcept;
    int GetStartIndex() const noexcept;

    T& operator[] (const int index);

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

    //скалярные
    TVector<T> operator*(const T& scalar);
    TVector<T> operator+(const T& scalar);
    TVector<T> operator-(const T& scalar);

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

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

    friend ostream& operator<<(ostream& out, const TVector<T>& v);
    friend istream& operator>>(istream& in, TVector<T>& v);
};
```

Назначение: представление вектора, у которого первые startIndex элементов нулевые(несуществующие).

Поля:

size – фактическая размерность вектора (сколько выделено памяти под элементы вектора).

startIndex – индекс первого «существующего» элемента вектора. Показывает, сколько нулевых(несуществующих) первых элементов в векторе.

pMem – память для представления элементов вектора.

Конструкторы:

```
TVector(int _size, int _startIndex);
```

Назначение: выделение и инициализация памяти объекта.

Входные параметры: **_size** – фактическая размерность вектора; **_startIndex** – индекс первого «существующего элемента».

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

Назначение: копирование данных из другого вектора.

Входные параметры: **v** – объект класса **TVector<T>**.

```
virtual ~TVector();
```

Назначение: очистка выделенной памяти.

Методы:

```
int GetSize() const noexcept;
```

Назначение: получение фактического размера вектора.

Выходные параметры: фактический размер вектора – количество выделенной памяти.

```
int GetStartIndex() const noexcept;
```

Назначение: получение индекса первого ненулевого(существующего) элемента вектора.

Выходные параметры: индекса первого ненулевого(существующего) элемента вектора.

```
T& operator[] (const int index);
```

Назначение: оператор индексации вектора, получение доступа к элементу вектора.

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

Выходные параметры: ссылка на элемент вектора.

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

Назначение: присваивание значений объекта **v**.

Входные параметры: **v** – объект класса **TVector<T>**.

Выходные параметры: ссылка на объект класса **TVector<T>**(себя).

`TVector<T> operator*(const T& scalar) ;`

Назначение: умножение вектора на скаляр.

Входные параметры: **`scalar`** – объект класса **`T`**.

Выходные параметры: объект класса **`TVector<T>`**.

`TVector<T> operator+(const T& scalar) ;`

Назначение: сложение вектора со скаляром.

Входные параметры: **`scalar`** – объект класса **`T`**.

Выходные параметры: объект класса **`TVector<T>`**.

`TVector<T> operator-(const T& scalar) ;`

Назначение: вычитание из вектора скаляр.

Входные параметры: **`scalar`** – объект класса **`T`**.

Выходные параметры: объект класса **`TVector<T>`**.

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

Назначение: сложение векторов.

Входные параметры: **`v`** – объект класса **`TVector<T>`**.

Выходные параметры: объект класса **`TVector<T>`**.

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

Назначение: вычитание векторов.

Входные параметры: **`v`** – объект класса **`TVector<T>`**.

Выходные параметры: объект класса **`TVector<T>`**.

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

Назначение: скалярное умножение векторов.

Входные параметры: **`v`** – объект класса **`TVector<T>`**.

Выходные параметры: объект класса **`T`**.

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

Назначение: проверка на равенство векторов.

Входные параметры: **v** – объект класса **TVector<T>**.

Выходные параметры: 1 или 0.

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

Назначение: проверка на неравенство векторов.

Входные параметры: **v** – объект класса **TVector<T>**.

Выходные параметры: 1 или 0.

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

Назначение: вывод данных вектора.

Входные параметры: **out** – поток вывода, **v** – объект класса **TVector<T>**.

Выходные параметры: поток вывода.

```
friend istream& operator>>(istream& in, TVector<T>& v);
```

Назначение: ввод данных вектора.

Входные параметры: **in** – поток ввода, **v** – объект класса **TVector<T>**.

Выходные параметры: поток ввода.

3.2.2 Описание класса **TMatrix**

```
template <typename T>
class TMatrix : public TVector<TVector<T>> {
public:
    TMatrix(int mn = 10);
    TMatrix(const TMatrix<T>& m);
    TMatrix(const TVector<TVector<T>>& m);

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

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

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

    TMatrix<T> operator+(const TMatrix<T>& m);
    TMatrix<T> operator-(const TMatrix<T>& m);

    TMatrix<T> operator*(const TMatrix<T>& m);
```

```

    friend ostream& operator<<(ostream& out, const TMatrix<T>& m);
    friend istream& operator>>(istream& in, TMatrix<T>& m);
};

```

Назначение: представление верхнетреугольной матрицы.

Конструкторы:

```
TMatrix(int mn);
```

Назначение: выделение и инициализация памяти объекта.

Входные параметры: **mn** – размерность верхнетреугольной матрицы.

```
TMatrix(const TMatrix<T>& m);
```

Назначение: копирование данных из другой матрицы.

Входные параметры: **m** – объект класса **TMatrix<T>**.

```
TMatrix(const TVector<TVector<T>>& m);
```

Назначение: копирование данных из матрицы типа **TVector<TVector<T>>**.

Входные параметры: **m** – объект класса **TVector<TVector<T>>**.

Методы:

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

Назначение: оператор индексации матрицы, получение доступа к вектору матрицы.

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

Выходные параметры: ссылка на элемент матрицы.

```
const TMatrix<T>& operator=(const TMatrix<T>& m);
```

Назначение: присваивание значений объекта **m**.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: ссылка на объект класса **TMatrix<T>**(себя).

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

Назначение: проверка на равенство матриц.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: 1 или 0.

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

Назначение: проверка на неравенство матриц.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: 1 или 0.

```
TMatrix<T> operator+(const TMatrix<T>& m);
```

Назначение: сложение матриц.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: объект класса **TMatrix<T>**.

```
TMatrix<T> operator-(const TMatrix<T>& m);
```

Назначение: вычитание матриц.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: объект класса **TMatrix<T>**.

```
TMatrix<T> operator*(const TMatrix<T>& m);
```

Назначение: умножение верхнетреугольных матриц.

Входные параметры: **m** – объект класса **TMatrix<T>**.

Выходные параметры: объект класса **TMatrix<T>**.

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

Назначение: вывод данных матрицы.

Входные параметры: **out** – поток вывода, **m** – объект класса **TMatrix<T>**.

Выходные параметры: поток вывода.

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

Назначение: ввод данных матрицы.

Входные параметры: **in** – поток ввода, **m** – объект класса **TMatrix<T>**.

Выходные параметры: поток ввода.

Заключение

В рамках работы был разработан шаблонный класс TVector для реализации вектора, поддерживающий операции: сложения, вычитания, умножение с скаляром и с другим вектором; доступ к элементу по индексу; ввод и вывод.

Также был разработан шаблонный класс TMatrix для реализации верхнетреугольной матрицы. Он поддерживает операции: сложение, вычитание, умножение с другой верхнетреугольной матрицей; доступ к строке и доступ к элементу по индексу; ввод и вывод.

Литература

1. Лекция «Вектора и матрицы» Сысоева А.В.
<https://cloud.unn.ru/s/wMPoPx2m9kPeyQz>
2. Определение умножения матриц
<https://ru.onlinemschool.com/math/library/matrix/multiply/>

Приложения

Приложение А. Реализация класса TVector

```
#ifndef _TVECTOR_H
#define _TVECTOR_H

#include <iostream>

using namespace std;

template <typename T>
class TVector {
protected:
    int size;
    int startIndex;
    T* pMem;
public:
    TVector(int _size = 10, int _startIndex = 0);
    TVector(const TVector<T>& v);
    virtual ~TVector();

    int GetSize() const noexcept;
    int GetStartIndex() const noexcept;

    T& operator[] (const int index);

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

    //скалярные
    TVector<T> operator*(const T& scalar);
    TVector<T> operator+(const T& scalar);
    TVector<T> operator-(const T& scalar);

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

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

    friend ostream& operator<<(ostream& out, const TVector<T>& v) {
        out << "| ";
        for (int i = 0; i < v.GetStartIndex(); i++)
            out << "0 ";
        for (int i = 0; i < v.GetSize(); i++)
            out << v.pMem[i] << " ";
        out << "| ";
        return out;
    }
    friend istream& operator>>(istream& in, TVector<T>& v) {
        for (int i = 0; i < v.GetSize(); i++)
            in >> v.pMem[i];
        return in;
    }
};

////////////////////////////////////

template <typename T>
TVector<T>::TVector(int _size, int _startIndex) {
```

```

        if (_size <= 0) throw "ERROR: Vector size less than 1";
        if (_startIndex < 0) throw "ERROR: Vector startIndex less than 0";

        size = _size;
        startIndex = _startIndex;
        pMem = new T[size];
    }
    template <typename T>
    TVector<T>::TVector(const TVector<T>& v) {
        size = v.size;
        startIndex = v.startIndex;
        pMem = new T[size];
        for (int i = 0; i < size; i++)
            pMem[i] = v.pMem[i];
    }
    template <typename T>
    TVector<T>::~~TVector() {
        if (size > 0)
            delete[] pMem;
    }

    template <typename T>
    int TVector<T>::GetSize() const noexcept {
        return size;
    }
    template <typename T>
    int TVector<T>::GetStartIndex() const noexcept {
        return startIndex;
    }

    template <typename T>
    T& TVector<T>::operator[](const int index) {
        if (index < 0 || index >= size + startIndex) throw "ERROR: index out of
range";

        if (index < startIndex) throw "ERROR: attempt to access a non-existent
vector element";
        return pMem[index - startIndex];
    }

    template <typename T>
    const TVector<T>& TVector<T>::operator=(const TVector<T>& v) {
        if (this == &v) return (*this);

        if (size != v.size) {
            delete[] pMem;
            size = v.size;
            pMem = new T[size];
        }

        startIndex = v.startIndex;
        for (int i = 0; i < size; i++)
            pMem[i] = v.pMem[i];

        return (*this);
    }

    //скалярные
    template <typename T>
    TVector<T> TVector<T>::operator*(const T& scalar) {
        TVector<T> res(*this);
        for (int i = 0; i < size; i++)

```

```

        res.pMem[i] *= scalar;
    return res;
}
template <typename T>
TVector<T> TVector<T>::operator+(const T& scalar) {
    TVector<T> res(*this);
    for (int i = 0; i < size; i++)
        res.pMem[i] += scalar;
    return res;
}
template <typename T>
TVector<T> TVector<T>::operator-(const T& scalar) {
    TVector<T> res(*this);
    for (int i = 0; i < size; i++)
        res.pMem[i] -= scalar;
    return res;
}

//векторные
template <typename T>
TVector<T> TVector<T>::operator+(const TVector<T>& v) {
    if (size != v.size) throw "ERROR: Vector diffirent size";
    if (startIndex != v.startIndex) throw "ERROR: Vector diffirent
startIndex";

    TVector<T> res(*this);
    for (int i = 0; i < size; i++)
        res.pMem[i] = res.pMem[i] + v.pMem[i];
    return res;
}
template <typename T>
TVector<T> TVector<T>::operator-(const TVector<T>& v) {
    if (size != v.size) throw "ERROR: Vector diffirent size";
    if (startIndex != v.startIndex) throw "ERROR: Vector diffirent
startIndex";

    TVector<T> res(*this);
    for (int i = 0; i < size; i++)
        res.pMem[i] = res.pMem[i] - v.pMem[i];
    return res;
}
template <typename T>
T TVector<T>::operator*(const TVector<T>& v) {
    if (size != v.size) throw "ERROR: Vector diffirent size";
    if (startIndex != v.startIndex) throw "ERROR: Vector diffirent
startIndex";

    T res = pMem[0] * v.pMem[0];
    for (int i = 1; i < size; i++)
        res = res + pMem[i] * v.pMem[i];
    return res;
}

template <typename T>
int TVector<T>::operator==(const TVector<T>& v) const {
    if (size != v.size || startIndex != v.startIndex) return 0;
    if (this == &v) return 1;

    int flag = 1;
    for (int i = 0; i < size; i++) {
        if (pMem[i] != v.pMem[i]) {
            flag = 0;
            break;
        }
    }
}

```

```

    }
    return flag;
}
template <typename T>
int TVector<T>::operator!=(const TVector<T>& v) const {
    return !(*this == v);
}

#endif // !_TVECTOR_H

```

Приложение Б. Реализация класса TMatrix

```

#ifndef _TMATRIX_H
#define _TMATRIX_H

#include "tvector.h"

template <typename T>
class TMatrix : public TVector<TVector<T>> {
public:
    TMatrix(int mn = 10);
    TMatrix(const TMatrix<T>& m);
    TMatrix(const TVector<TVector<T>>& m);

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

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

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

    TMatrix<T> operator+(const TMatrix<T>& m);
    TMatrix<T> operator-(const TMatrix<T>& m);

    TMatrix<T> operator*(const TMatrix<T>& m);

    friend ostream& operator<<(ostream& out, const TMatrix<T>& m) {
        for (int i = 0; i < m.GetSize(); i++) {
            out << m.pMem[i] << endl;
        }
        return out;
    }
    friend istream& operator>>(istream& in, TMatrix<T>& m) {
        for (int i = 0; i < m.GetSize(); i++) {
            in >> m.pMem[i];
        }
        return in;
    }
};

////////////////////////////////////

template <typename T>
TMatrix<T>::TMatrix(int mn) : TVector<TVector<T>>(mn) {
    if (mn <= 0) throw "ERROR: matrix size less than 1";
    for (int i = 0; i < mn; i++) {
        pMem[i] = TVector<T>(mn - i, i);
    }
}

template <typename T>
TMatrix<T>::TMatrix(const TMatrix<T>& m) : TVector<TVector<T>>((TVector<TVector<T>>)m) {}

```

```

template <typename T>
TMatrix<T>::TMatrix(const TVector<TVector<T>>& m) : TVector<TVector<T>>(m) {}

template <typename T>
TVector<T>& TMatrix<T>::operator[](const int index) {
    return TVector<TVector<T>>::operator[](index);
}

template <typename T>
const TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& m) {
    return TVector<TVector<T>>::operator=(m);
}

template <typename T>
int TMatrix<T>::operator==(const TMatrix<T>& m) const {
    return TVector<TVector<T>>::operator==(m);
}

template <typename T>
int TMatrix<T>::operator!=(const TMatrix<T>& m) const {
    return !(*this == m);
}

template <typename T>
TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& m) {
    return TVector<TVector<T>>::operator+(m);
}

template <typename T>
TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& m) {
    return TVector<TVector<T>>::operator-(m);
}

template <typename T>
TMatrix<T> TMatrix<T>::operator*(const TMatrix<T>& m) {
    if (size != m.size) throw "ERROR: Matrix diffirent size";
    if (startIndex != m.startIndex) throw "ERROR: Matrix diffirent
startIndex";

    TMatrix<T> res(size);
    for (int i = 0; i < size; i++)
        for (int j = res[i].GetStartIndex(); j < size; j++)
            res[i][j] = (*this)[i][j] - (*this)[i][j];

    for (int i = 0; i < size; i++)
        for (int j = res[i].GetStartIndex(); j < size; j++)
            for (int k = res[i].GetStartIndex(); k <= j; k++)
                res[i][j] += (*this)[i][k] * m.pMem[k][j];
    return res;
}

#endif // !_MATRIX_H

```