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:

**«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](#_Toc153647330)

[1 Постановка задачи 4](#_Toc153647331)

[2 Руководство пользователя 5](#_Toc153647332)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc153647333)

[2.2 Приложение для демонстрации работы матриц 6](#_Toc153647334)

[3 Руководство программиста 8](#_Toc153647335)

[3.1 Описание алгоритмов 8](#_Toc153647336)

[3.1.1 Векторы 8](#_Toc153647337)

[3.1.2 Матрицы 10](#_Toc153647338)

[3.2 Описание программной реализации 12](#_Toc153647339)

[3.2.1 Описание класса TVector 12](#_Toc153647340)

[3.2.2 Описание класса TMatrix 15](#_Toc153647341)

[Заключение 18](#_Toc153647342)

[Литература 19](#_Toc153647343)

[Приложения 20](#_Toc153647344)

[Приложение А. Реализация класса TVector 20](#_Toc153647345)

[Приложение Б. Реализация класса TMatrix 23](#_Toc153647346)

# 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.

# 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.

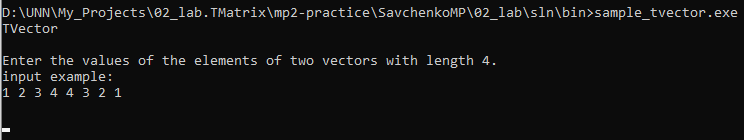
Задачи при реализации класса TMatrix:

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.

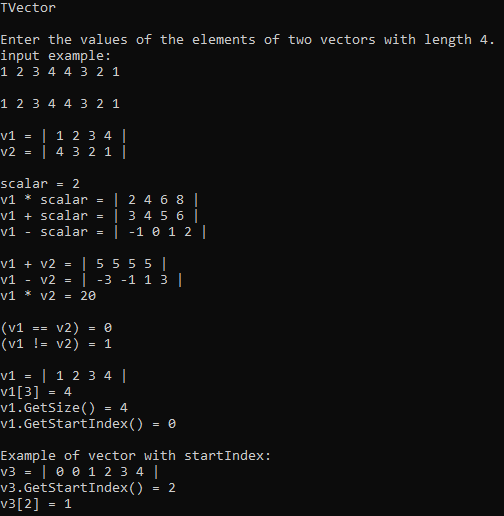
# Руководство пользователя

## Приложение для демонстрации работы векторов

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



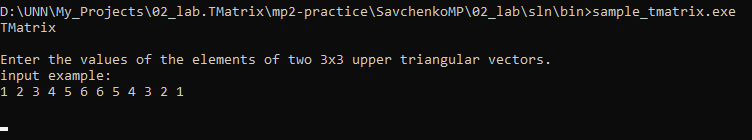
1. Основное окно программы sample\_tvector.exe
2. Введите значения координат двух четырехмерных векторов, например, «1 2 3 4 3 3 2 1». После ввода в окне напечатаются новые данные, показанные ниже ().



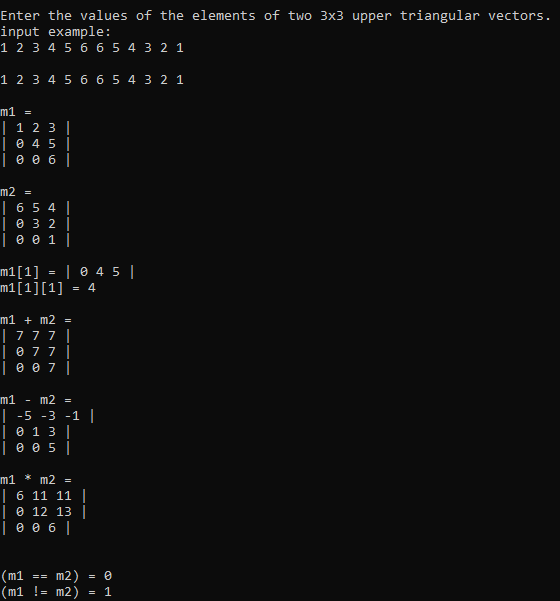
1. Выведенными данные программы sample\_tvector.exe
2. Здесь вы можете ознакомиться с основными операциями над векторами: сложение, вычитание, умножение со скаляром; сложение, вычитание с вектором; скалярное произведение; пример вектора с ненулевым начальным индексом.

## Приложение для демонстрации работы матриц

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



1. Основное окно программы sample\_tmatrix.exe
2. 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 ().



1. Выведенные данные программы sample\_tmatrix.exe
2. 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.

# Руководство программиста

## Описание алгоритмов

### Векторы

Вектор представляет собой структуру данных, используемую для хранения элементов одного и того же типа. Обозначим вектор как , где m – количество элементов в векторе (размер вектора); n – количество первых нулевых (нейтральных) элементов, которые всегда будут таковыми являться (далее такие элементы будем называть их нулевыми); m-n – количество обычных элементов, которые могут принимать разные значения, включая нулевые (нейтральные).

Операции, которые можно производить только с векторами с одинаковым размером и одинаковым количеством нулевых элементов:

* Векторно-скалярные операции: включают в себя сложение, вычитание и умножение вектора на элемент того же типа, что хранится в векторе.
* 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. Эти операции не затрагивают первые n нулевые элементы вектора .

Пример:

Пример:

**Векторно-векторные операции**

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

Пример:

Пример:

**Скалярное произведение**

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

Пример:

**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.

### Матрицы

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

Верхнетреугольную матрицу можно записать так вектор её строк:

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.

В случае верхнетреугольной матрицы эти операции можно представить как поэлементное сложение или вычитание векторов строк матрицы:

Пример:

**Умножение матриц**

Результатом умножения матриц и будет матрица такая, что элемент матрицы C, стоящий в i-той строке и j-том столбце (), равен сумме произведений элементов i-той строки матрицы A на соответствующие элементы j-того столбца матрицы B:

В случае, когда матрицы и – верхнетреугольные, то при их умножении можно не производить операции, связанные с умножением на элемент ниже главной диагонали, можно не производить, т.к. такой элемент всегда нулевой. Поэтому приведенная выше формула для матрицы примет следующий вид:

Пример:

**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.

## Описание программной реализации

### 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.

### 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 // !\_TMATRIX\_H