

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:
**«СТРУКТУРЫ ХРАНЕНИЯ ДЛЯ МАТРИЦ
СПЕЦИАЛЬНОГО ВИДА»**

Выполнил(а): студент(ка) группы
3822Б1ФИ1

_____ / Стариков Н.В./
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	7
3 Руководство программиста	10
3.1 Используемые алгоритмы	10
3.1.1 Вектор.....	10
3.1.2 Матрица.....	10
3.2 Описание классов.....	13
3.2.1 Класс TVector.....	13
3.2.2 Класс TMartix.....	13
Заключение	20
Литературы	21
Приложения	22
Приложение А. Реализация класса TVector	22
Приложение Б. Реализация класса TMartix	24

Введение

Целью лабораторной работы является изучение и практическое применение концепции шаблонов на примере треугольных матриц и их представления в виде вектора, состоящего из векторов. Для реализации этой программы понадобится создать классы с шаблонами и различными функциями. Шаблоны позволяют создавать обобщенные типы данных, которые могут быть использованы с различными типами данных без необходимости дублирования кода. Треугольная матрица — в линейной алгебре квадратная матрица, у которой все элементы, стоящие ниже (или выше) главной диагонали, равны нулю. Служат для более компактного хранения данных.

1 Постановка задачи

Цель: реализация классов для представления вектора TVector и матрицы TMatrix как вектора векторов.

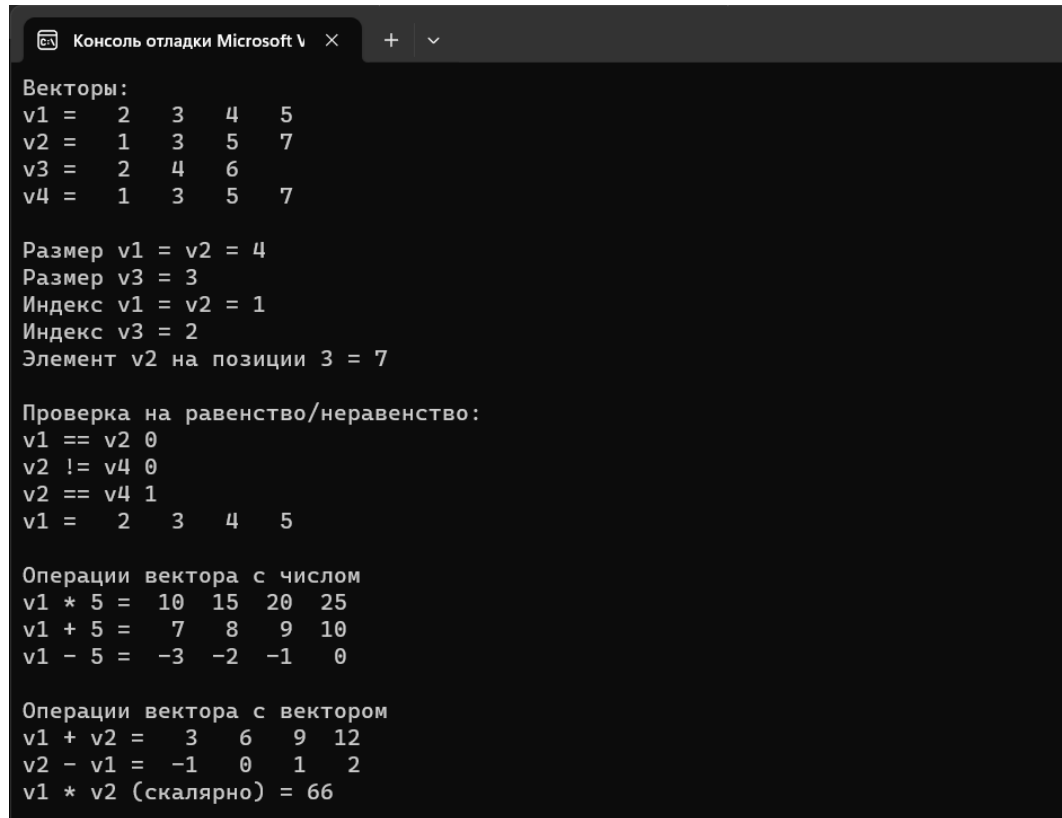
Задачи:

1. Изучить основные принципы работы шаблонов.
2. Разработать шаблонный класс, поддерживающий основные операции, для реализации вектора.
3. Разработать шаблонный класс, поддерживающий основные операции, для реализации матрицы.
4. Провести тестирования разработанных шаблонных классов.
5. Обеспечить работоспособность тестов, покрывающих все методы классов TMatrix и TVector.
6. Сделать выводы о проделанной работе.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

1. Запустить `sample_tvector.exe`. В результате появится следующее окно (рис. 1).



```
Консоль отладки Microsoft V x + v
Векторы:
v1 = 2 3 4 5
v2 = 1 3 5 7
v3 = 2 4 6
v4 = 1 3 5 7

Размер v1 = v2 = 4
Размер v3 = 3
Индекс v1 = v2 = 1
Индекс v3 = 2
Элемент v2 на позиции 3 = 7

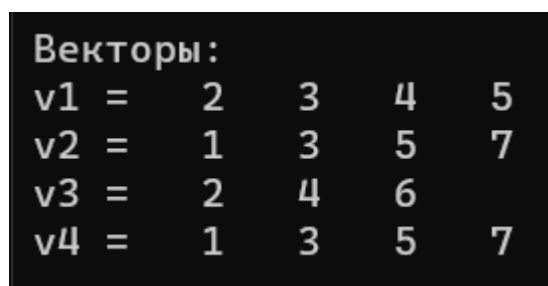
Проверка на равенство/неравенство:
v1 == v2 0
v2 != v4 0
v2 == v4 1
v1 = 2 3 4 5

Операции вектора с числом
v1 * 5 = 10 15 20 25
v1 + 5 = 7 8 9 10
v1 - 5 = -3 -2 -1 0

Операции вектора с вектором
v1 + v2 = 3 6 9 12
v2 - v1 = -1 0 1 2
v1 * v2 (скалярно) = 66
```

Рис. 1. Основное окно программы

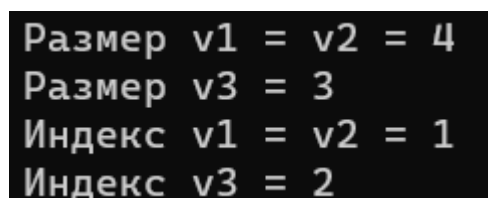
2. Создаётся четыре вектора (рис. 2).



```
Векторы:
v1 = 2 3 4 5
v2 = 1 3 5 7
v3 = 2 4 6
v4 = 1 3 5 7
```

Рис. 2. Создание векторов

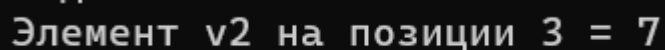
3. Далее вызывается метод для получения размера и начальных индексов векторов (рис. 3).



```
Размер v1 = v2 = 4
Размер v3 = 3
Индекс v1 = v2 = 1
Индекс v3 = 2
```

Рис. 3. Получение размера и начальных индексов векторов

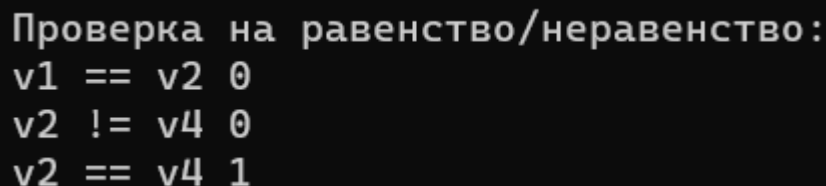
4. Затем выполняем операцию получения значения третьего элемента вектора v2 (рис. 4).



```
Элемент v2 на позиции 3 = 7
```

Рис. 4. Результат работы программы

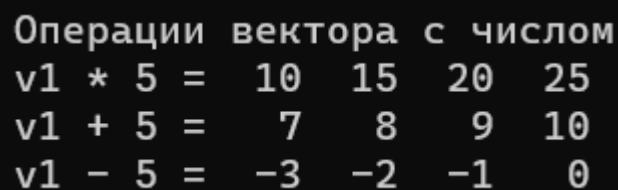
5. Следующим шагом сравниваем вектора (рис. 5).



```
Проверка на равенство/неравенство:  
v1 == v2 0  
v2 != v4 0  
v2 == v4 1
```

Рис. 5. Операции сравнения векторов

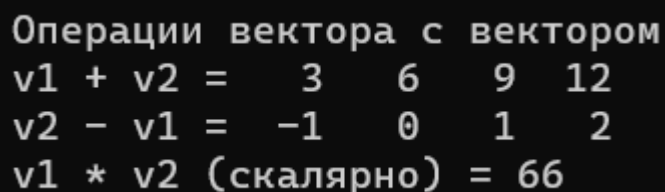
6. Далее выполняются операции умножения, сложения и вычитания вектора и числа (рис. 6).



```
Операции вектора с числом  
v1 * 5 = 10 15 20 25  
v1 + 5 = 7 8 9 10  
v1 - 5 = -3 -2 -1 0
```

Рис. 6. Операции вектора с числом

7. И в конце выполняются операции сложения и вычитания векторов, а также их скалярное произведение (рис. 7).



```
Операции вектора с вектором  
v1 + v2 = 3 6 9 12  
v2 - v1 = -1 0 1 2  
v1 * v2 (скалярно) = 66
```

Рис. 7. Операции вектора с вектором

2.2 Приложение для демонстрации работы матриц

1. Запустить sample_tmatrix.exe. В результате появится следующее окно (рис. 8).

```

Матрицы:
A:
  1  1  1
  0  2  2
  0  0  3

B:
  2  2  2
  0  4  4
  0  0  6

C:
  1  1  1
  0  2  2
  0  0  3

Операции:
A + B:
  3  3  3
  0  6  6
  0  0  9

A - B:
 -1 -1 -1
  0 -2 -2
  0  0 -3

A * B:
  2  6  12
  0  8  20
  0  0  18

Проверка на равенство/неравенство:
A == B  0
A != B  1
A == C  1

```

Рис. 8. Основное окно программы

2. Сначала создаются матрицы A, B, C (рис. 9).

```

Матрицы:
A:
  1  1  1
  0  2  2
  0  0  3

B:
  2  2  2
  0  4  4
  0  0  6

C:
  1  1  1
  0  2  2
  0  0  3

```

Рис. 9. Матрицы

3. Затем выполняются операции сложения, вычитания и умножения матриц (рис. 10).

Операции:			
A + B:			
3	3	3	
0	6	6	
0	0	9	
A - B:			
-1	-1	-1	
0	-2	-2	
0	0	-3	
A * B:			
2	6	12	
0	8	20	
0	0	18	

Рис. 10. Операции над матрицами

4. Последним шагом мы сравниваем матрицы (рис. 11).

Проверка на равенство/неравенство:			
A == B	0		
A != B	1		
A == C	1		

Рис. 11. Операции сравнения матриц

3 Руководство программиста

3.1 Используемые алгоритмы

3.1.1 Вектор

Вектор хранится в виде указателя на массив элементов одного типа данных, стартового индекса и количества элементов в векторе. Такая структура позволяет эффективно работать с матричными операциями. Если стартовый индекс отличен от нуля, то все элементы от 0 до стартового индекса будут равны нейтральному элементу типа данных.

Вектор поддерживает операции сложения, вычитания и умножения с элементом типа данных, сложения, вычитания, скалярного произведения с вектором того же типа данных, операции индексации, сравнение на равенство (неравенство).

I. Операция сложения

Операция сложения определена следующим образом: складываются элементы первого и второго вектора с одинаковыми индексами, или каждый элемент вектора отдельно складывается с элементом.

Пример:

Сложение векторов:

$$v1 = \{2, 3, 4, 5\}$$

$$v2 = \{1, 3, 5, 7\}$$

$$v1 + v2 = \{2, 3, 4, 5\} + \{1, 3, 5, 7\} = \{2+1, 3+3, 4+5, 5+7\} = \{3, 6, 9, 12\}$$

Сложение вектора с константой, равной 5:

$$v1 = \{2, 3, 4, 5\}$$

$$v1 + 5 = \{2, 3, 4, 5\} + 5 = \{2+5, 3+5, 4+5, 5+5\} = \{7, 8, 9, 10\}$$

II. Операция вычитания

Операция вычитания определена следующим образом: вычитаются элементы первого и второго вектора с одинаковыми индексами, или каждый элемент вектора отдельно вычитается с элементом.

Пример:

Вычитание векторов:

$$v1 = \{2, 3, 4, 5\}$$

$$v2 = \{1, 3, 5, 7\}$$

$$v1 - v2 = \{2, 3, 4, 5\} - \{1, 3, 5, 7\} = \{2-1, 3-3, 4-5, 5-7\} = \{1, 0, -1, -2\}$$

Вычитание из вектора константы, равной 5:

$$v1 = \{2, 3, 4, 5\}$$

$$v1 - 5 = \{2, 3, 4, 5\} - 5 = \{2-5, 3-5, 4-5, 5-5\} = \{-3, -2, -1, 0\}$$

III. Операция умножения

Операция умножения определена следующим образом: скалярное произведение векторов, или каждый элемент вектора отдельно умножается с элементом.

Пример:

Скалярное произведение векторов:

$$v2 = \{1, 3, 5, 7\}$$

$$v1 = \{2, 3, 4, 5\}$$

$$v2 * v1 = \{1, 3, 5, 7\} * \{2, 3, 4, 5\} = 1*2 + 3*3 + 5*4 + 7*5 = 66$$

Произведение вектора с константой, равной 5:

$$v1 * 5 = \{2, 3, 4, 5\} * 5 = \{2*5, 3*5, 4*5, 5*5\} = \{10, 15, 20, 25\}$$

IV. Операция индексации

Операция индексации предназначена для получения элемента вектора.

Пример:

$$v1 = \{2, 3, 4, 5\}.$$

$$\text{Получение индекса 0: } v1[0] = 2$$

$$\text{Получение индекса 1: } v1[1] = 3$$

V. Операция сравнения на равенство

Операция сравнения на равенство с вектором возвращает 1, если все элементы векторов равны, причём их стартовые индексы и размеры тоже равны. В противном случае возвращает 0.

$$v1 = \{2, 3, 4, 5\}$$

$$v2 = \{1, 3, 5, 7\}$$

$$v4 = \{1, 3, 5, 7\}$$

$$\text{Сравнение векторов } v1 \text{ с } v2: (v1 == v2) = 0$$

$$\text{Сравнение векторов } v2 \text{ с } v3: (v2 == v3) = 1$$

VI. Операция сравнения на неравенство

Операция сравнения на равенство с вектором возвращает 0, если все элементы векторов равны, причём их стартовые индексы и размеры тоже равны, 1 в противном случае.

$$v2 = \{1, 3, 5, 7\}$$

$$v3 = \{1, 3, 5, 7\}$$

$$\text{Сравнение векторов } v2 \text{ с } v3: (v2 != v3) = 0$$

3.1.2 Матрица

Матрица хранится в виде указателя на указатели на массивы элементов одного типа данных, стартового индекса и количества элементов в матрице (именно количество столбцов или строк, так как матрица квадратная и верхнетреугольная).

Матрица поддерживает следующие операции:

I. Операция сложения

Операция сложения определена для матрицы следующим образом: складываются элементы первой и второй матрицы с одинаковыми индексами.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

$$A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{pmatrix} = \begin{pmatrix} 3 & 3 & 3 \\ 0 & 6 & 6 \\ 0 & 0 & 9 \end{pmatrix}$$

II. Операция вычитания

Операция вычитания определена для матрицы следующим образом: вычитаются элементы первой и второй матрицы с одинаковыми индексами.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$A - B = \begin{pmatrix} a_{11} - b_{11} & a_{12} - b_{12} & a_{13} - b_{13} \\ a_{21} - b_{21} & a_{22} - b_{22} & a_{23} - b_{23} \\ a_{31} - b_{31} & a_{32} - b_{32} & a_{33} - b_{33} \end{pmatrix} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & -2 & -2 \\ 0 & 0 & -3 \end{pmatrix}$$

III. Операция умножения

Операция умножения определена для матрицы того же типа.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$A * B = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} 2 & 6 & 12 \\ 0 & 8 & 20 \\ 0 & 0 & 18 \end{pmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31} = 1*2 + 1*0 + 1*0 = 2$$

$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32} = 1*2 + 1*4 + 1*0 = 6$$

$$c_{13} = a_{11} * b_{13} + a_{12} * b_{23} + a_{13} * b_{33} = 1*2 + 1*4 + 1*6 = 12$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} = 0*2 + 2*0 + 2*0 = 0$$

$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32} = 0*2 + 2*4 + 2*0 = 8$$

$$c_{23} = a_{21} * b_{13} + a_{22} * b_{23} + a_{23} * b_{33} = 0*2 + 2*4 + 2*6 = 20$$

$$c_{31} = a_{31} * b_{11} + a_{32} * b_{21} + a_{33} * b_{31} = 0*2 + 0*0 + 3*0 = 0$$

$$c_{32} = a_{31} * b_{12} + a_{32} * b_{22} + a_{33} * b_{32} = 0*2 + 0*4 + 3*0 = 0$$

$$c_{33} = a_{31} * b_{13} + a_{32} * b_{23} + a_{33} * b_{33} = 0*2 + 0*4 + 3*6 = 18$$

IV. Операция индексации

Операция индексации предназначена для получения элемента матрицы.

Элемент матрицы – вектор-строка.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$A[0] = \{1, 1, 1\}$$

$$A[2][2] = 3$$

V. Операция сравнения на равенство

Операция сравнения на равенство с матрицей возвращает 1, если все элементы матриц равны, причём их стартовые индексы и размеры тоже равны. В противном случае возвращает 0.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

$$C = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

$$(A == B) = 0$$

$$(A == C) = 1$$

VI. Операция сравнения на неравенство

Операция сравнения на неравенство с матрицей возвращает 0, если все элементы матриц равны, причём их стартовые индексы и размеры тоже равны.

В противном случае возвращает 0.

Пример:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

$$(A \neq B) = 1$$

3.2 Описание классов

3.2.1 Класс TVector

Объявление класса:

```
template <class T>
class TVector
{
protected:
    int size;
    int start_index;
    T* pVec;
public:
    TVector(int s = 10, int index = 0);
    TVector(const TVector<T>& vec);
    ~TVector();
    int GetSize() const;
    int GetIndex() const;

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

    TVector operator* (const T& v);
    TVector operator+ (const T& v);
    TVector operator- (const T& v);

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

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

    template<typename T> friend std::ostream& operator<<(std::ostream& ostr, const
TVector<T>& v);
    template<typename T> friend std::istream& operator>>(std::istream& istr, TVector<T>&
v);
};
```

Назначение: представление вектора.

Поля:

Size – количество элементов вектора.

Start_Index – индекс первого необходимого элемента вектора.

***pVec** – память для представления элементов вектора.

Методы:

TVector(int s = 10, int index = 0);

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **s** – длина вектора, **index** – стартовый индекс.

TVector(const TVector<T>& vec);

Назначение: конструктор копирования.

Входные параметры: **vec** – экземпляр класса

~TVector();

Назначение: освобождение выделенной памяти.

int GetSize() const;

Назначение: получение размера вектора.

Выходные параметры: количество элементов вектора.

int GetIndex() const;

Назначение: получение стартового индекса.

Выходные параметры: стартовый индекс.

Операции:

T& operator[] (const int index);

Назначение: перегрузка операции индексации.

Входные параметры: **index** – индекс (позиция) элемента.

Выходные параметры: элемент, который находится на **index** позиции.

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

Назначение: оператор сравнения.

Входные параметры: **v** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если не равны, 1 – если равны.

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

Назначение: оператор сравнения.

Входные параметры: **v** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если равны, 1 – если не равны.

TVector operator*(const T& v);

Назначение: оператор умножения вектора на значение.

Входные параметры: **v** – элемент, на который умножаем вектор.

Выходные параметры: экземпляр класса, элементы которого в **v** раз больше.

TVector operator+(const T& v);

Назначение: оператор сложения вектора и значения.

Входные параметры: **v** – элемент, с которым складываем вектор.

Выходные параметры: экземпляр класса, элементы которого на **v** больше.

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

Назначение: оператор вычитания вектора и значения.

Входные параметры: **v** – элемент, который вычитаем из вектора.

Выходные параметры: экземпляр класса, элементы которого на **v** меньше.

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

Назначение: оператор сложения векторов.

Входные параметры: **v** – вектор, который суммируем.

Выходные параметры: экземпляр класса, равный сумме двух векторов.

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

Назначение: оператор умножения векторов.

Входные параметры: **v** – вектор, на который умножаем.

Выходные параметры: значение, равное скалярному произведению двух векторов.

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

Назначение: оператор разности двух векторов.

Входные параметры: **v** – вектор, который вычитаем.

Выходные параметры: экземпляр класса, равный разности двух векторов.

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

Назначение: оператор присваивания.

Входные параметры: **v** – экземпляр класса, который присваиваем.

Выходные параметры: ссылка на (***this**), уже присвоенный экземпляр класса.

```
template<typename T> friend std::ostream& operator<<(std::ostream& ostr,  
const TVector<T>& v);
```

Назначение: оператор ввода вектора.

Входные параметры: **istr** – поток ввода, **v** – ссылка на вектор, который вводим.

Выходные параметры: поток ввода.

```
template<typename T> friend std::istream& operator>>(std::istream& istr,  
TVector<T>& v);
```

Назначение: оператор вывода вектора.

Входные параметры: **ostr** – поток вывода, **v** – ссылка на вектор, который выводим.

Выходные параметры: поток вывода.

3.2.2 Класс TMatrix

Объявление класса:

```
template <typename T> class TMatrix : public TVector <TVector<T>>  
{  
public:
```

```

TMatrix(int mn = 10);
TMatrix(const TMatrix& m);
TMatrix(const TVector<TVector<T>>& m);

const TMatrix operator=(const TMatrix& m);
int operator==(const TMatrix& m) const;
int operator!=(const TMatrix& m) const;

TMatrix operator+(const TMatrix& m);
TMatrix operator-(const TMatrix& m);
TMatrix operator*(const TMatrix& m);

template<typename T> friend std::ostream& operator<<(std::ostream& ostr, const
TMatrix<T>& v);
template<typename T> friend std::istream& operator>>(std::istream& istr, TMatrix<T>&
v);
};

```

Поля:

Size – размерность матрицы.

Start_Index – индекс первого необходимого элемента.

***pVec** – память для представления элементов матрицы.

Методы:

```
TMatrix(int mn = 10);
```

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **mn** – длина вектора (по умолчанию 10).

```
TMatrix(const TMatrix& m);
```

Назначение: конструктор копирования.

Входные параметры: **m** – экземпляр класса.

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

Назначение: конструктор преобразования типов.

Входные параметры: **m** – ссылка на **TVector<TVector<T>>** - на объект, который преобразуем.

Операторы:

```
const TMatrix operator=(const TMatrix& m);
```

Назначение: оператор присваивания.

Входные параметры: **m** – экземпляр класса, который присваиваем.

Выходные параметры: ссылка на (***this**), уже присвоенный экземпляр класса.

```
int operator==(const TMatrix& m) const;
```

Назначение: оператор сравнения.

Входные параметры: **m** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если не равны, 1 – если равны.

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

Назначение: оператор сравнения.

Входные параметры: **m** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если равны, 1 – если не равны.


```
TMatrix operator+(const TMatrix& m);
```

Назначение: оператор сложения матриц.

Входные параметры: **m** – матрица, которую суммируем.

Выходные параметры: экземпляр класса, равный сумме двух матриц.

```
TMatrix operator-(const TMatrix& m);
```

Назначение: оператор вычитания матриц.

Входные параметры: **m** – матрица, которую вычитаем.

Выходные параметры: экземпляр класса, равный разности двух матриц.

```
TMatrix operator*(const TMatrix& m);
```

Назначение: оператор умножения матриц.

Входные параметры: **m** – матрица, которую умножаем.

Выходные параметры: экземпляр класса, равный произведению двух матриц.

```
template<typename T> friend std::ostream& operator<<(std::ostream& ostr,  
const TMatrix<T>& v);
```

Назначение: оператор ввода матрицы.

Входные параметры: **istr** – поток ввода, **v** – ссылка на матрицу.

Выходные параметры: поток ввода.

```
template<typename T> friend std::istream& operator>>(std::istream& istr,  
TMatrix<T>& v);
```

Назначение: оператор вывода матрицы.

Входные параметры: **ostr** – поток вывода, **v** – ссылка на матрицу.

Выходные параметры: поток вывода.

Заключение

В ходе выполнения работы мы изучили и практически применили концепцию шаблонов. Мы разработали шаблонный класс для реализации вектора, который поддерживает основные операции, такие как: сложение, вычитание и умножение с элементом типа данных, сложение, вычитание, скалярного произведения с вектором того же типа данных, операции индексации, сравнение на равенство (неравенство).

Также мы разработали шаблонный класс для реализации матрицы, который поддерживает операции сложения матриц, вычитания матриц, умножения матрицы на матрицу, операции индексации, сравнение на равенство (неравенство).

Литературы

1. Треугольная матрица [https://ru.wikipedia.org/wiki/Треугольная_матрица].
2. Лекция «Вектора и матрицы» [<https://cloud.unn.ru/s/FkYBW5rJLDCgBmJ>].

Приложения

Приложение А. Реализация класса TVector

```
// конструкторы
template <typename T>
TVector<T> ::TVector(int s, int index)
{
    if (s < 0)
        throw - 1;
    else
        if (s == 0)
        {
            size = s;
            pVec = NULL;
        }
        else
        {
            start_index = index;
            size = s;
            pVec = new T[size];
            for (int i = 0; i < size; i++)
                pVec[i] = 0;
        }
}

template <typename T>
TVector<T> ::TVector(const TVector<T>& vec)
{
    size = vec.size;
    start_index = vec.start_index;
    pVec = new T[size];
    for (int i = 0; i < size; i++)
    {
        pVec[i] = vec.pVec[i];
    }
}

// деструктор
template <typename T>
TVector<T>::~TVector()
{
    if (size > 0)
    {
        size = 0;
        delete[] pVec;
        pVec = NULL;
    }
}

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

template <typename T>
int TVector<T> ::GetIndex() const
{
    return start_index;
}

//операторы
template <typename T>
T& TVector<T> :: operator [] (const int index)
{
    if (index < 0 || index >= size)
```

```

        throw "Индекс не может быть отрицательным или превышать размер";
    else
        return pVec[index];
}

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

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

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

// операции вектора с числом

//вектор * число
template <typename T>
TVector<T> TVector<T>::operator*(const T& v)
{
    TVector<T> res(size);
    for (int i = 0; i < size; i++)
        res[i] = (*this)[i] * v;
    return res;
}

// вектор + константа
template <typename T>
TVector<T> TVector<T>::operator+(const T& n)
{
    TVector<T> res(size);
    for (int i = 0; i < size; i++)
        res[i] = (*this)[i] + n;
    return res;
}

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

//операции вектора с вектором

// вектор + вектор
template <typename T>
TVector<T> TVector<T>:: operator+(const TVector<T>& v)
{
    if (size != v.size)
        throw "Не удастся собрать векторы с разными измерениями";

    if (start_index != v.start_index)
        throw "Не удастся собрать векторы с разными индексами";

    TVector<T> tmp(*this);

```

```

        for (int i = 0; i < size; i++)
        {
            tmp.pVec[i] = pVec[i] + v.pVec[i];
        }
        return tmp;
    }
    // вектор - вектор
    template <typename T>
    TVector<T> TVector<T>::operator-(const TVector<T>& v)
    {
        if (size != v.size)
            throw "Не удастся вычесть векторы с разными размерами";

        if (start_index != v.start_index)
            throw "Не удастся вычесть векторы с разными индексами";

        TVector<T> tmp(*this);
        for (int i = 0; i < size; i++)
        {
            tmp.pVec[i] = pVec[i] - v.pVec[i];
        }
        return tmp;
    }
    // скалярное умножение
    template <typename T>
    T TVector<T>::operator*(const TVector<T>& v)
    {
        if (size != v.size)
            throw "Нельзя скалярно умножать векторы разного размера";

        if (start_index != v.start_index)
            throw "Нельзя ли скалярно умножать векторы с разными индексами";

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

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

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

        start_index = v.start_index;
        for (int i = 0; i < size; i++)
        {
            pVec[i] = v.pVec[i];
        }
        return *this;
    }

    // вывод
    template <typename T>
    std::ostream& operator<<(std::ostream& ostr, const TVector<T>& v)
    {
        for (int i = 0; i < v.size; i++)
            ostr << std::setw(3) << v.pVec[i] << " ";
        return ostr;
    }

```

```

}
// ввод
template <typename T>
std::istream& operator>>(std::istream& istr, TVector<T>& v)
{
    istr >> v.size;
    std::cout << "\nВведите " << v.size << " координаты: ";
    for (int i = 0; i < v.size; i++)
        istr >> v.pVec[i];
    return istr;
}
#endif

```

Приложение Б. Реализация класса TMatrix

```
//конструкторы
template <typename T>
TMatrix<T>::TMatrix<T>(int mn) :TVector<TVector<T>>(mn)
{
    for (int i = 0; i < mn; i++)
    {
        pVec[i] = TVector<T>(mn - i, i);
    }
}

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

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

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

// A + B
template<typename T>
TMatrix<T> TMatrix<T>::operator+(const TMatrix& m)
{
    if (size != m.size)
        throw "Не удастся собрать матрицу с разными размерами";
    else
        return TVector<TVector<T>>::operator+(m);
}

// A - B
template<typename T>
TMatrix<T> TMatrix<T>::operator-(const TMatrix& m)
{
    if (size != m.size)
        throw "Не удастся вычесть матрицу с разными размерами";
    else
        return TVector<TVector<T>>::operator-(m);
}

// A * B
template<typename T>
TMatrix<T> TMatrix<T>::operator*(const TMatrix& m)
{
    if (size != m.size)
        throw "Не удастся умножить матрицу с разными размерами";
    else
        size;
    TMatrix <T> res(size);
```



```

        for (int i = 0; i < size; i++)
            for (int j = i; j < size; j++)
            {
                for (int k = i; k <= j; k++)
                    res.pVec[i][j - i] += this->pVec[i][k - i] * m.pVec[k][j -
k];
            }
        return res;
    }

// ввод
template <typename T>
std::istream& operator>>(std::istream& istr, TMatrix<T>& m)
{
    istr >> m.size;
    std::cout << "\nВведите " << m.size << " элементы: ";
    for (int i = 0; i < m.size; i++)
        istr >> m.pVec[i];
    return istr;
}

// вывод
template <typename T>
std::ostream& operator<<(std::ostream& ostr, const TMatrix<T>& m)
{
    for (int i = 0; i < m.size; i++)
    {
        for (int j = 0; j < m.pVec[i].GetIndex(); j++)
            ostr << std::setw(3) << "0" << " ";
        ostr << m.pVec[i] << std::endl;
    }
    return ostr;
}
#endif

```