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:

«Структуры хранения для матриц специального вида»

Выполнил: студент группы 3822Б1ФИ2
_____ / Коробейников А.П.
Подпись

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

Нижний Новгород
2023

Содержание

Введение

В данной лабораторной работе мы будем рассматривать матрицы особо вида, а именно - треугольные. Треугольная матрица — в линейной алгебре квадратная матрица, у которой все элементы, стоящие ниже (или выше) главной диагонали, равны нулю. Представлять треугольные матрицы мы будем в виде вектора, которой состоит из векторов действительных чисел. Для работы с такими матрицами, создадим библиотеку, в которой будет реализован шаблонный класс векторов и класс матриц (класс наследник от класса векторов). Эта работа актуальна, поскольку треугольные матрицы играют важную роль линейной алгебре и соответственно в современном мире, поскольку линейна алгебра используется во многих областях науки и нужна для решения множества задач.

1 Постановка задачи

Цель – разработать библиотеку, которая содержит в себе структуру хранения для шаблонных векторов и верхнетреугольных матриц (в виде вектора из векторов действительных чисел).

Задачи:

1. Изучить теорию линейной алгебры о векторах и матрицах, чтобы можно было правильно реализовать арифметические операции.
2. Изучить способы реализации матрицы через вектора и выбрать подходящий.
3. Написать класс шаблонного вектора.
4. Написать тесты для класса векторов, проверяющие корректность работы каждого метода.
5. Написать класс матриц.
6. Написать тесты для класса матриц, проверяющие корректность работы каждого метода.
7. Написать программу демонстрирующую работу класса матриц.
8. Написать программу демонстрирующую работу класса векторов.

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).

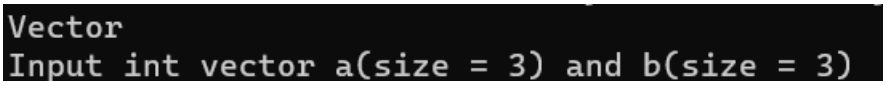

Рис. 1. Начальное окно программы

2. Введите два вектора a и b, как в примере на Рис. 2.

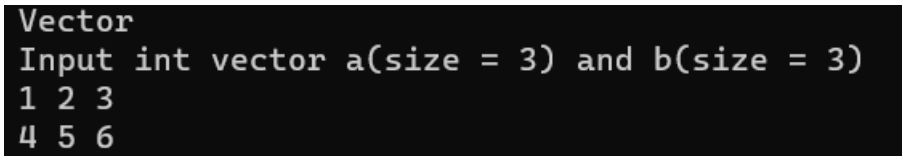

Рис. 2. Ввод векторов

3. В результате появится окно, показанное ниже (Рис. 3).

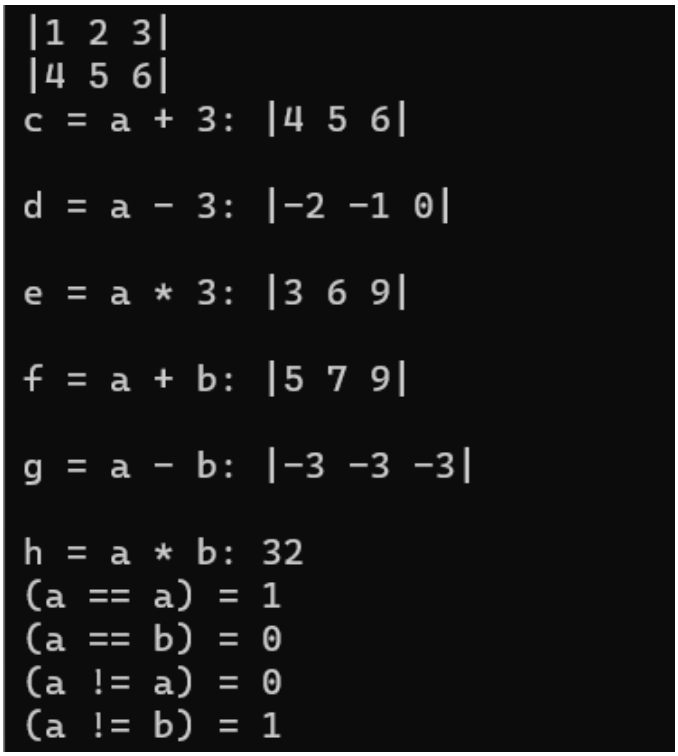

Рис. 3. Основное окно программы

4. Сначала выводятся вектора, которые вы ввели. Затем демонстрируется работа всех операций (предусмотренных в классе векторов) с векторами a и b.

2.2 Приложение для демонстрации работы матриц

1. Запустите приложение с названием sample_tmatrix.exe. В результате появится окно, показанное ниже (Рис. 4).

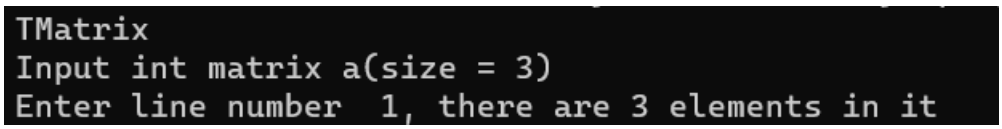

Рис. 4. Начальное окно программы

2. Вам следует ввести матрицу a, следуя инструкциям появляющимся в окне. Пример приведён на Рис. 5.

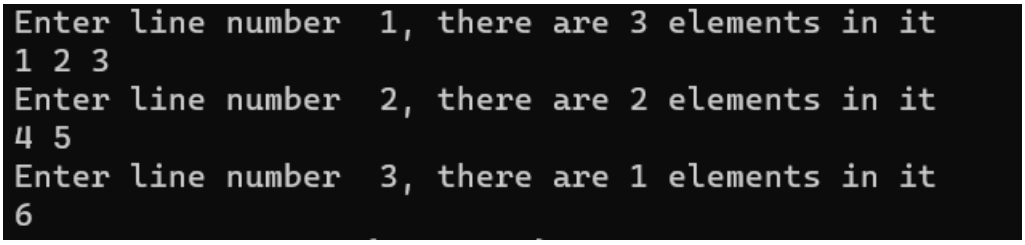

Рис. 5. Пример ввода матрицы a

3. Затем точно также введите матрицу b. Пример приведён на Рис. 6 .

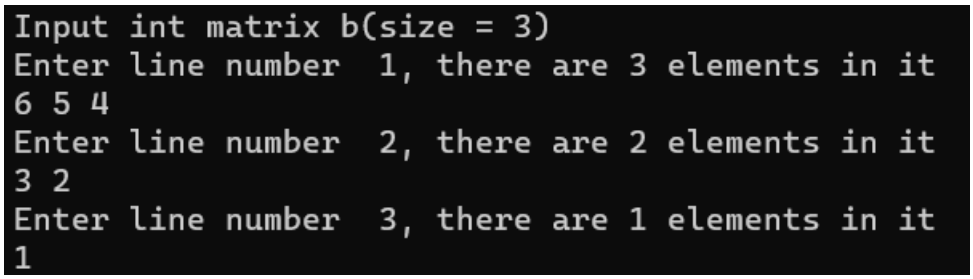

Рис. 6. Пример ввода матрицы b

4. Сначала в окне выведутся матрицы, которые вы ввели (Рис. 7).

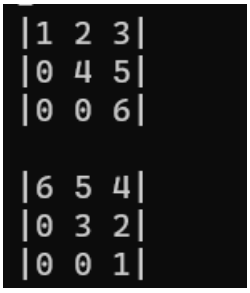

Рис. 7. Операции сравнения матриц

5. А также будет продемонстрирована работа всех операций (предусмотренных в классе матриц) с матрицами a и b (Рис. 8).

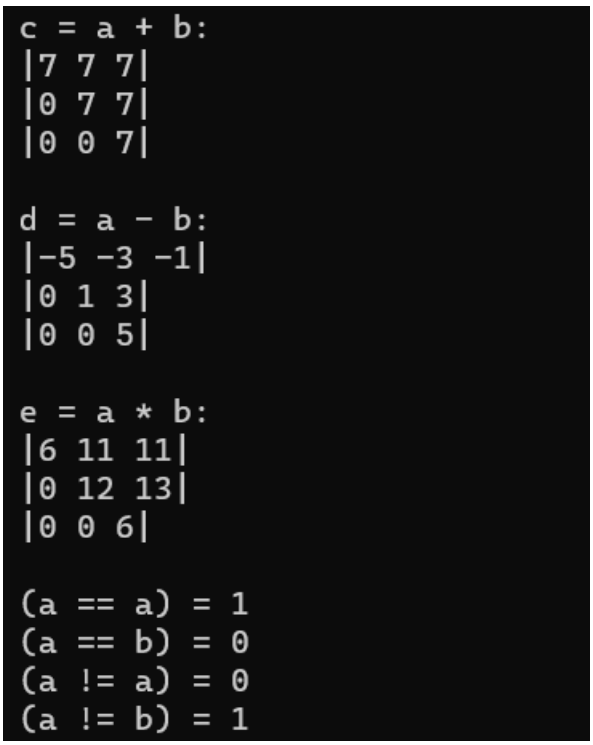

Рис. 8. Основное окно программы

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Вектор

Вектор – структура хранения. Он хранит элементы одного типа данных.

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).

Операция сложения

Операция сложения определена для вектора того же типа (складываются элементы первого и второго вектора с одинаковыми индексами) или некоторого элемента того же типа (каждый элемент вектора отдельно складывается с элементом).

Пример:

Сложение векторов v1 = {1, 2, 3, 4} и v2 = {1, 3, 5, 7}

v1 + v2 = {2, 5, 8, 11}

Сложение вектора v1, с константой равной 5

v1 + 5 = {6, 7, 8, 11}

Операция вычитания

Операция вычитания определена для вектора того же типа (вычитаются элементы первого и второго вектора с одинаковыми индексами) или некоторого элемента того же типа (каждый элемент вектора отдельно вычитается с элементом).

Пример:

Разность векторов v2 = {1, 3, 5, 7} и v1 = {1, 2, 3, 4}

v2 – v1 = {0, 1, 2, 3}

Вычитание из вектора v1 константы, равной 5

v1 - 5 = {-4, -3, -2, -1}

Операция умножения

Операция умножения определена для вектора того же типа (скалярное произведение векторов) или некоторого элемента того же типа (каждый элемент вектора отдельно умножается с элементом).

Пример:

Скалярное произведение векторов v2 = {1, 3, 5, 7} и v1 = {1, 2, 3, 4}

v2 * v1 = 50

Произведение вектора v1 с константой, равной 5

v1 * 5 = {5, 10, 15, 20}

Операция индексации

Операция индексации предназначена для получения элемента вектора. Причем, если позиция будет меньше, чем стартовый индекс, то будет выведен нейтральный элемент для данного типа данных.

Пример:

v1 = {1, 2, 3, 4}. Получение индекса 1 и 0 соответственно v1[1] = 2, v1[0] = 1

Операция сравнения на равенство

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

Пример:

v1 = {1, 2, 3, 4}, v2 = {1, 3, 5, 7}, v3 = {1, 3, 5, 7}

Сравнение векторов v1 с v2 и v2 с v3

(v1 == v2)? 0

(v2 == v3)? 1

Операция сравнения на неравенство

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

v2 = {1, 3, 5, 7}, v3 = {1, 3, 5, 7}

Сравнение векторов v2 с v3

(v2 != v3)? 0

3.1.2 Матрица

Матрица – вектор векторов, структура хранения. Она хранит элементы одного типа данных.

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 операции сложения, вычитания и умножения с матрицей того же типа данных, операции индексации, сравнение на равенство (неравенство).

Операция сложения

Операция сложения определена для матрицы того же типа (складываются элементы первой и второй матрицы с одинаковыми индексами).

Пример: $A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$

$A + B = \begin{pmatrix} 3 & 3 & 3 \\ 0 & 6 & 6 \\ 0 & 0 & 9 \end{pmatrix}$

Операция вычитания

Операция вычитания определена для матрицы того же типа (вычитаются элементы первой и второй матрицы с одинаковыми индексами).

Пример: $A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$

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

Операция умножения

Операция умножения определена для матрицы того же типа.

Пример: $A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 4 \\ 0 & 0 & 6 \end{pmatrix}$

A * B = $\begin{pmatrix} 2 & 6 & 12 \\ 0 & 8 & 20 \\ 0 & 0 & 18 \end{pmatrix}$

Операция сравнения на равенство

Операция сравнения на равенство с матрицей возвращает 1, если они равны поэлементно, причём их стартовые индексы и размеры тоже равны, 0 в противном случае.

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

(B == C)? 1

Операция сравнения на неравенство

Операция сравнения на неравенство с матрицей возвращает 0, если они равны поэлементно, причём их стартовые индексы и размеры тоже равны, 1 в противном случае.

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)? 1

Операция индексации

Операция индексации предназначена для получения элемента матрицы.

Элемент матрицы – вектор-строка, также можно вывести элемент матрицы по индексу, так как для вектора также перегружена операция индексации.

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

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

A[1][1] = 2

3.2 Описание программной реализации

3.2.1 Описание класса TVector

```
template <typename T>
class Vector {
protected:
      int n;
      T* coor;
      int start_index;
public:
      Vector(int size= 10, int start_index = 0);
      Vector(const Vector<T>& obj);
      ~Vector();
      int GetSize() const noexcept;
      int GetStart_index() const noexcept;
      T& operator[] (const int ind);
      T& operator[] (const int ind) const;
      Vector <T> operator + (const Vector<T>& obj);
```

```
      Vector <T> operator - (const Vector<T>& obj);
      double operator * (const Vector<T>& obj);
      Vector <T> operator + (const T& obj);
      Vector <T> operator - (const T& obj);
      Vector <T> operator * (const T& obj);
      double length() const;
      bool operator == (const Vector<T>& obj) const;
      bool operator != (const Vector<T>& obj) const; const          Vector<T>&
operator = (const Vector<T>& obj);

      friend std::ostream& operator << (std::ostream& stream, const Vector<T>&
obj);
      friend std::istream& operator >> (std::istream& stream, Vector<T>& obj);
};
```

Назначение: представление вектора.

Поля:

`n` – количество элементов вектора.

`*coor` – память для представления элементов вектора.

`start_Index` – индекс первого необходимого элемента вектора.

Методы:

```
TVector(int size = 10, int start_index = 0);
```

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: `size` – длина вектора, `start_index` – стартовый индекс.

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

Назначение: конструктор копирования.

Входные параметры: `obj` – экземпляр класса, на основе которого создаем новый объект.

```
~TVector();
```

Назначение: освобождение выделенной памяти.

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

Назначение: получение размера вектора.

Выходные параметры: количество элементов вектора.

```
int GetStart_index() const;
```

Назначение: получение стартового индекса.

Выходные параметры: стартовый индекс.

Операции:

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

Назначение: перегрузка операции индексации.

Входные параметры: `ind` – индекс (позиция) элемента.

Выходные параметры: элемент, который находится на `ind` позиции.

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

Назначение: константная перегрузка операции индексации.

Входные параметры: `ind` – индекс (позиция) элемента.

Выходные параметры: элемент, который находится на `ind` позиции.

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

Назначение: оператор сравнения.

Входные параметры: `obj` – экземпляр класса, с которым сравниваем.

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

```
bool operator!=(const TVector<T>& obj) const;
```

Назначение: оператор сравнения.

Входные параметры: `obj` – экземпляр класса, с которым сравниваем.

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

```
TVector operator*(const T& obj);
```

Назначение: оператор умножения вектора на значение.

Входные параметры: `obj` – элемент, на который умножаем вектор.

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

```
TVector operator+(const T& obj);
```

Назначение: оператор сложения вектора и значения.

Входные параметры: `obj` – элемент, с которым складываем вектор.

Выходные параметры: экземпляр класса, элементы которого на obj больше.

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

Назначение: оператор вычитания вектора и значения.

Входные параметры: `obj` – элемент, который вычитаем из вектора.

Выходные параметры: экземпляр класса, элементы которого на obj меньше.

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

Назначение: оператор сложения векторов.

Входные параметры: `obj` – вектор, который суммируем.

Выходные параметры: экземпляр класса, равный сумме двух векторов.

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

Назначение: оператор умножения векторов.

Входные параметры: `obj` – вектор, на который умножаем.

Выходные параметры: значение, равное скалярному произведению двух векторов.

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

Назначение: оператор разности двух векторов.

Входные параметры: `obj` – вектор, который вычитаем.

Выходные параметры: экземпляр класса, равный разности двух векторов.

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

Назначение: оператор присваивания.

Входные параметры `obj` – экземпляр класса, который присваиваем.

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

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

Назначение: оператор ввода вектора.

Входные параметры: `istr` – поток ввода, `obj` – ссылка на вектор, который вводим.

Выходные параметры: поток ввода.

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

Назначение: оператор вывода вектора.

Входные параметры: `ostr` – поток вывода, `obj` – ссылка на вектор, который выводим.

Выходные параметры: поток вывода.

3.2.2 Описание класса TMatrix

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

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

  bool operator == (const TMatrix<T>& m) const;
  bool 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 istream& operator>>(istream& istr, TMatrix<T>& m);
  friend ostream& operator<<(ostream& ostr, const TMatrix<T>& m);
};
```

Класс наследуется (тип наследования pubic) от класса `Vector<Vector<T>>`

Назначение: представление матрицы как вектор векторов.

Поля:

`n` – размерность матрицы.

`Start_Index` – индекс первого необходимого элемента.

`*coor` – 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, на основе которого создаём новый объект.

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

Назначение: конструктор преобразования типов.

Входные параметры: `m` – ссылка на `Vector<Vector<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::istream& operator>>(std::istream& istr,
TMatrix<T>& m);
```
Назначение: оператор ввода матрицы.

Входные параметры: `istr` – поток ввода, `m` – ссылка на матрицу, которую вводим.

Выходные параметры: поток ввода.

```
template<typename T> friend std::ostream& operator<<(std::ostream& ostr,
const TMatrix<T>& m);
```
Назначение: оператор вывода матрицы.

Входные параметры: `ostr` – поток вывода, `m` – ссылка на матрицу, которую выводим.

Выходные параметры: поток вывода.

Заключение

В ходе выполнения лабораторной работы мы выполнили поставленные нами задачи. А именно мы разработали шаблонный класс векторов, которые выполняет все необходимые операции. Написали тесты и убедились, что всё работает корректно. Затем мы разработали класс матриц, мы сделали это через публичное наследование от вектора из векторов действительных чисел и также написали тесты. Были написаны приложения, демонстрирующие работу классов векторов и матриц.

Литература

1. Лекция «Вектора и матрицы» Сысоев А.В [https://cloud.unn.ru/s/FkYBW5rJLDCgBmJ].

Приложения

Приложение А. Реализация класса TVector

```
#include <iostream>
using namespace std;

// конструкторы

template <typename T>
Vector<T>::Vector(int size, int start_index) {
	if (size < 0) {
		throw "negative length";
	}
	if (start_index < 0) {
		throw "negative start_index";
	}
	n = size;
	this->start_index = start_index;
	coor = new T[size];
}

template <typename T>
Vector<T>::Vector(const Vector<T>& obj) {
	n = obj.n;
	start_index = obj.start_index;
	coor = new T[n];
	for (int i = 0; i < n; i++) {
		coor[i] = obj.coor[i];
	}
}

//деструктор
template <typename T>
Vector<T>::~Vector() {
	n = 0;
	this->start_index = 0;
	delete[] coor;
}

// Get
template <typename T>
int Vector<T>::GetSize() const noexcept {
	return n;
}

template <typename T>
int Vector<T>::GetStart_index() const noexcept {
	return (start_index);
}

// длина вектора
template <typename T>
double Vector<T>::length() const {
	double res = 0.0;
	for (int i = 0; i < n; i++) {
		res += coor[i] * coor[i];
	}
	return (sqrt(res));
}
```

```
// перегрузка операций

template <typename T>
T& Vector<T>:: operator[] (const int ind)  //индексация
{
	return (coor[ind]);
}

template <typename T>
T& Vector<T>:: operator[] (const int ind) const  //индексация
{
	return (coor[ind]);
}


template <typename T>
Vector<T> Vector<T>:: operator + (const Vector<T>& obj) // сложение векторов
{
	if (n != obj.n) {
		throw "Incorrect data (different size)";
	}
	if (start_index != obj.start_index) {
		throw "Incorrect data (different start_index)";
	}

	Vector<T> tmp(*this);
	for (int i = 0; i < n; i++) {
		tmp.coor[i] = tmp.coor[i] + obj.coor[i];
	}
	return tmp;
}

template <typename T>
Vector <T> Vector<T>:: operator - (const Vector<T>& obj) //вычитание векторов
{
	if (n != obj.n) {
		throw "Incorrect data (different size)";
	}
	if (start_index != obj.start_index) {
		throw "Incorrect data (different start_index)";
	}
	Vector<T> tmp(*this);
	for (int i = 0; i < n; i++) {
		tmp.coor[i] = tmp.coor[i] - obj.coor[i];
	}
	return tmp;
}



template <typename T>
double Vector<T>:: operator * (const Vector<T>& obj) //умножение векторов
{
	if (n != obj.n) {
		throw "Incorrect data (different size)";
	}
	if (start_index != obj.start_index) {
		throw "Incorrect data (different start_index)";
	}
	Vector<T> tmp(*this);
	double res = 0.0;
	for (int i = 0; i < n; i++) {
		res += tmp.coor[i] * obj.coor[i];
```

```
	}
	return res;
}


template <typename T>
Vector<T> Vector<T>:: operator + (const T& obj) //сложение вектора и T
{
	Vector<T> tmp(*this);
	for (int i = 0; i < n; i++) {
		tmp.coor[i] += obj;
	}
	return tmp;
}

template <typename T>
Vector<T> Vector<T>:: operator - (const T& obj) //вычитание из вектора  T
{
	Vector<T> tmp(*this);
	for (int i = 0; i < n; i++) {
		tmp.coor[i] -= obj;
	}
	return tmp;
}

template <typename T>
Vector<T> Vector<T>:: operator * (const T& obj) //умножение вектора и T
{
	Vector<T> tmp(*this);
	for (int i = 0; i < n; i++) {
		tmp.coor[i] *= obj;
	}
	return tmp;
}


template <typename T>
bool Vector<T>:: operator == (const Vector<T>& obj) const     //операция
проверки на равенство
{
	if (n != obj.n) {
		return false;
	}
	if (start_index != obj.start_index) {
		return false;
	}
	for (int i = 0; i < n; i++) {
		if (coor[i] != obj.coor[i]) {
			return false;
		}
	}
	return true;
}

template <typename T>
bool Vector<T>::operator != (const Vector<T>& obj) const //операция проверки
на неравенство
{
	if (n != obj.n) {
		return true;
	}
	if (start_index != obj.start_index) {
		return true;
```

```
	}
	for (int i = 0; i < n; i++) {
		if (coor[i] != obj.coor[i]) {
			return true;
		}
	}
	return false;
}



template <typename T>
const Vector<T>& Vector<T>::operator = (const Vector<T>& obj)   //оператор
присваивания
{
	if (this == &obj) {
		return(*this);
	}
	if (n != obj.n) {
		delete[] coor;
		n = obj.n;
		coor = new T[n];
	}

	start_index = obj.start_index;

	for (int i = 0; i < n; i++) {
		coor[i] = obj.coor[i];
	}

	return (*this);
}
```

Приложение Б. Реализация класса TMatrix

```
#include "tvector.h"


// конструкторы

template <typename T>
TMatrix<T>::TMatrix(int mn): Vector<Vector<T>>(mn) {
	for (int i = 0; i < mn; ++i) {
		coor[i] = Vector<T>(mn - i, i);
	}
}

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

template <typename T>
TMatrix<T>::TMatrix<T>(const Vector<Vector<T>>& v) : Vector<Vector<T>> (v) {}

//операторы

template <typename T>
bool TMatrix<T>:: operator == (const TMatrix<T>& m) const //проверка на
равенство
{
	return Vector<Vector<T>> :: operator == (m);
}
```

```
template <typename T>
bool TMatrix<T>:: operator != (const TMatrix<T>& m) const //проверка на неравенство
{
	return Vector<Vector<T>> :: operator != (m);
}

template <class T>
const TMatrix<T>& TMatrix<T>::operator = (const TMatrix<T>& m) //оператор присваивания
{
	return Vector<Vector<T>>::operator=(m);
}

template <typename T>
TMatrix<T> TMatrix<T>:: operator + (const TMatrix<T>& m) //сложение матриц
{
	return Vector<Vector<T>> :: operator + (m);
}

template <typename T>
TMatrix<T> TMatrix<T>:: operator - (const TMatrix<T>& m) //вычитание матриц
{
	return Vector<Vector<T>> :: operator - (m);
}

template <typename T>
TMatrix<T> TMatrix<T>:: operator * (const TMatrix<T>& m) {
	if (n != m.n) {
		throw "No multi (different size)";
	}
	TMatrix res(n);

	for (int i = 0; i < n; ++i) {
		for (int j = i; j < n; ++j) {
				res[i][j - i] = 0;
		}
	}

	for (int i = 0; i < n; ++i) {
		for (int j = i; j < n; ++j) {
			for (int k = i; k <= j; ++k) {
				res[i][j - i] += (*this)[i][k-i] * m[k][j - k];
			}
		}
	}


	return res;
}
```