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:

«ВЕКТОРА И МАТРИЦЫ»

Выполнил: студент группы
3822Б1ФИ1
_____ / Созонов И.С. /
Подпись

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

Нижний Новгород
2023

Содержание

Введение

Вектор – это понятие из линейной алгебры, объект, имеющий длину и направление. Проще всего его описать как направленный отрезок. Он может обозначаться графически или на записи – стрелкой или числом. В аналитике и разработке вектор также понимают как упорядоченный набор чисел.

Векторы нужны для описания реальных и абстрактных сущностей: скорости, действия силы на предмет и так далее. Все эти сущности объединяет наличие размера и направления. С помощью векторов их можно описывать полно или подробно.

Вектор на плоскости или в пространстве – в любой системе координат – можно выразить как набор координат. Этот набор будет максимально точно его описывать. Поэтому вектор можно представить как упорядоченный набор чисел, и этим активно пользуются разработчики, аналитики и другие специалисты. В этом смысле вектор – вроде линии из чисел, и его можно использовать как структуру для хранения данных. Упорядоченный набор векторов представляет собой матрицу – упорядоченный набор чисел или элементов, организованных в виде прямоугольной таблицы.

Матричные обозначения широко распространены в современной математике и её приложениях. Матрица – полезный аппарат для исследования многих задач теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений.

Следствием разнообразия областей применения матричного аппарата в современной науке является наличие в любом из больших математических программных комплексов (Mathcad, Mathematica, Derive, Mapple) подсистем, выполняющих операции над матрицами, а также существование специальных программных библиотек (ScalaPack, PlaPack), рассчитанных на обработку огромных (десятки и сотни тысяч строк) матриц, в том числе с использованием распределенных (параллельных) вычислений.

Помимо матриц общего вида, для которых наиболее естественной и наиболее часто используемой представляется программная реализация в виде двумерного массива, в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные и другие). В данной лабораторной работе рассматриваются верхнетреугольные матрицы. Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры, и потому более эффективных.

1 Постановка задачи

Цель – реализовать классы для представления векторов TVector и верхнетреугльных матриц TMatrix.

Задачи:

1. Разработать класс TVector для работы с векторами. Написать следующие операции для работы с векторами: вычисление длины вектора, сравнение векторов, сложение вектора со скаляром, разность вектора со скаляром, умножение вектора на скаляр, сложение векторов, разность векторов, скалярное произведение векторов.
2. Разработать класс TMatrix для работы с верхнетреугольными матрицами. Написать следующие операции для работы с векторами: сравнение матриц, сложение матриц, разность матриц, умножение матриц.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

1. Запустить sample_tvector.exe. В результате появится окно для ввода длины вектора a (рис. 1).

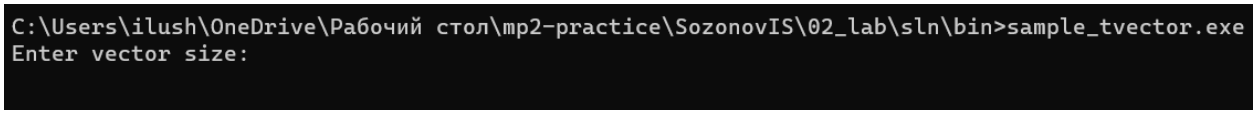

Рис. 1. Основное окно приложения

2. Ввести длину вектора. В результате появится окно для ввода вектора a (Рис. 2).

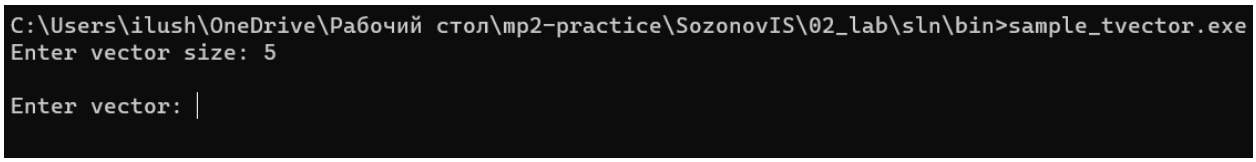

Рис. 2. Ввод длины вектора a

3. Ввести вектор a. В результате появится окно для ввода длины вектора b (Рис. 3).

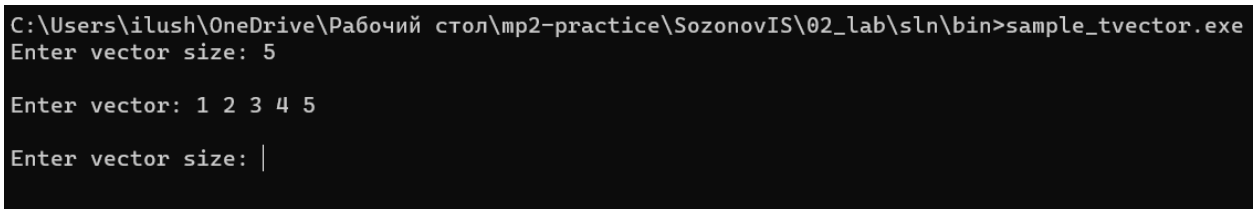

Рис. 3. Ввод вектора a

4. Ввести длину вектора b. В результате появится окно для ввода вектора b (Рис. 4).

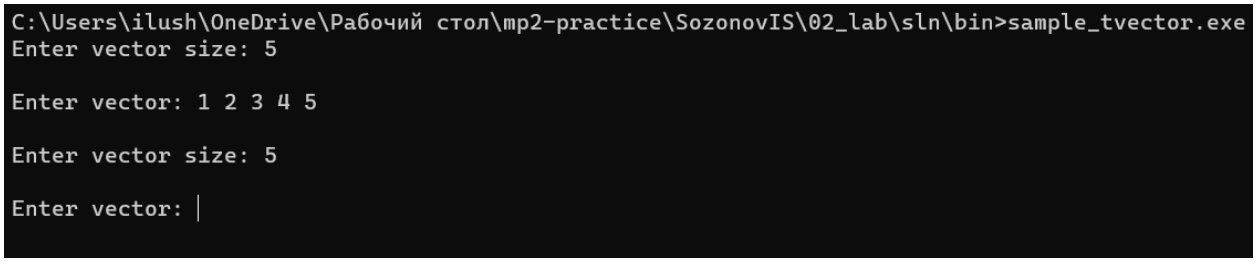

Рис. 4. Ввод длины вектора b

5. Ввести вектор b. В результате будут выведены вектор a и вектор b, результаты их сравнения, сложения, разности и скалярного произведения. Появится окно для ввода случайного числа (Рис. 5).

```
C:\Users\ilush\OneDrive\Рабочий стол\mp2-practice\SozonovIS\02_lab\sln\bin>sample_tvector.exe
Enter vector size: 5

Enter vector: 1 2 3 4 5

Enter vector size: 5

Enter vector: -7 9 4 -3 2

Vector a: 1 2 3 4 5
Vector b: -7 9 4 -3 2

a == b ? 0
a != b ? 1

a + b: -6 11 7 1 7
a - b: 8 -7 -1 7 3
a * b: 21

Enter a random number:
```

Рис. 5. Ввод вектора b

6. Ввести случайное число. В результате будут выведены результаты сложения вектора a с введенным числом, разности вектора b с введенным числом и умножения вектора a на введенное число (Рис. 6).

```
C:\Users\ilush\OneDrive\Рабочий стол\mp2-practice\SozonovIS\02_lab\sln\bin>sample_tvector.exe
Enter vector size: 5

Enter vector: 1 2 3 4 5

Enter vector size: 5

Enter vector: -7 9 4 -3 2

Vector a: 1 2 3 4 5
Vector b: -7 9 4 -3 2

a == b ? 0
a != b ? 1

a + b: -6 11 7 1 7
a - b: 8 -7 -1 7 3
a * b: 21

Enter a random number: 6

a + 6: 7 8 9 10 11
b - 6: -13 3 -2 -9 -4
a * 6: 6 12 18 24 30

C:\Users\ilush\OneDrive\Рабочий стол\mp2-practice\SozonovIS\02_lab\sln\bin>
```

Рис. 6. Ввод случайного числа

2.2 Приложение для демонстрации работы верхнетреугольных матриц

1. Запустить sample_tmatrix.exe. В результате появится окно для ввода размера верхнетреугольной матрицы A (Рис. 7).

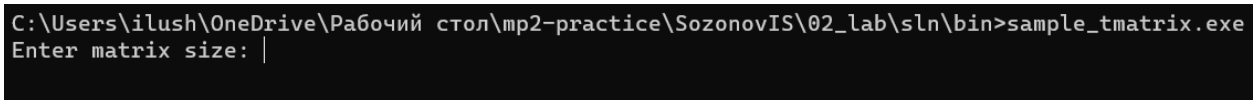

Рис. 7. Основное окно приложения

2. Ввести размер матрицы A. В результате появится окно для ввода векторов матрицы A (**Ошибка! Источник ссылки не найден.**).

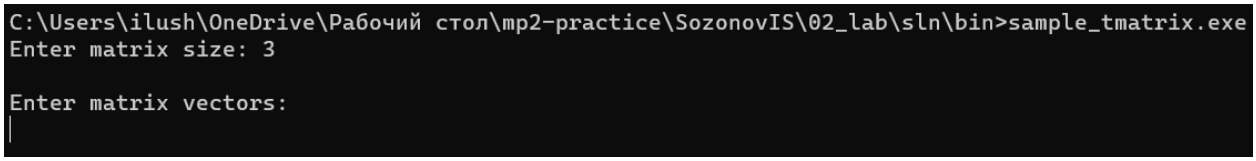

Рис. 8. Ввод размера матрицы A

3. Ввести векторы матрицы A. В результате появится окно для ввода размера верхнетреугольной матрицы B (**Ошибка! Источник ссылки не найден.**).

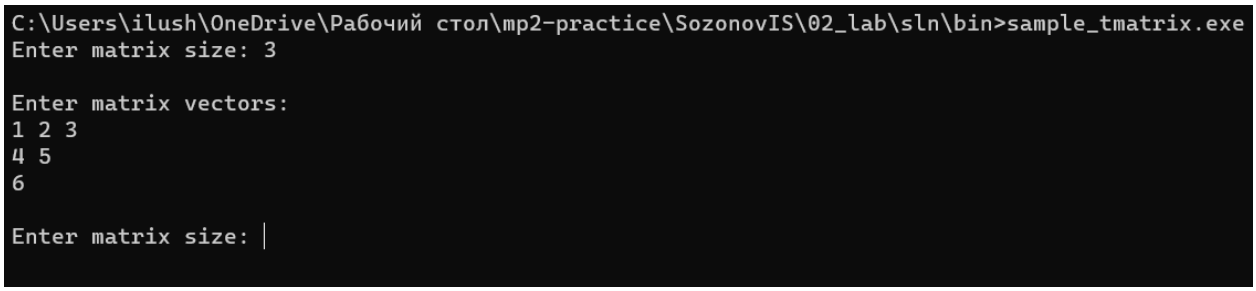

Рис. 9. Ввод векторов матрицы A

4. Ввести размер матрицы B. В результате появится окно для ввода векторов матрицы B (**Ошибка! Источник ссылки не найден.**).

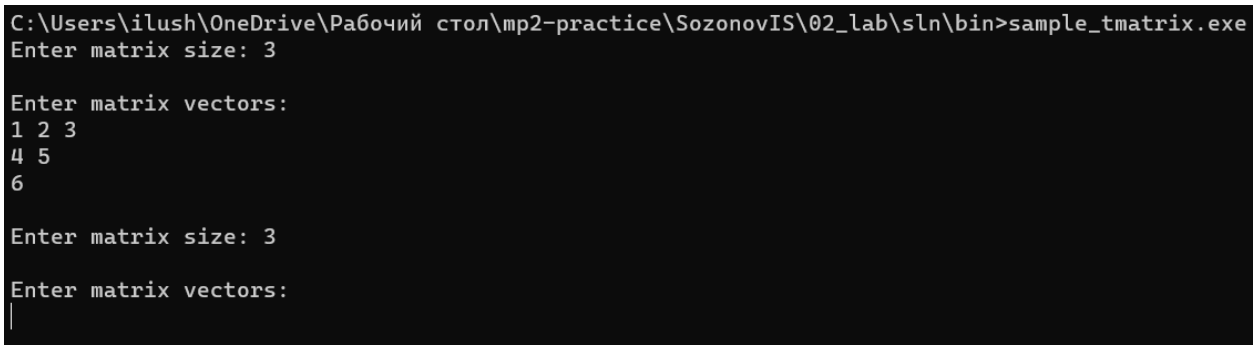

Рис. 10. Ввод размера матрицы B

5. Ввести векторы матрицы B. В результате будут выведены верхнетреугольные матрицы A и B, результаты их сравнения, сложения, разности и умножения (**Ошибка! Источник ссылки не найден.**).

```
Enter matrix size: 3

Enter matrix vectors:
-7 1 9
5 -2
8

Matrix A:
1 2 3
0 4 5
0 0 6

Matrix B:
-7 1 9
0 5 -2
0 0 8


A == B ? 0
A != B ? 1

A + B:
-6 3 12
0 9 3
0 0 14

A - B:
8 1 -6
0 -1 7
0 0 -2

A * B:
-7 11 29
0 20 32
0 0 48


C:\Users\ilush\OneDrive\Рабочий стол\mp2-practice\SozonovIS\02_lab\sln\bin>
```

Рис. 11. Ввод мощности второго множества

3 Руководство программиста

3.1 Использованные алгоритмы

3.1.1 Векторы

Вектор – в математике – набор v_i (чисел, математических выражений), состоящий из n элементов.

Структура данных вектора $v = (v_1, v_2, \dots, v_n)$ есть

$S_v = (M_v p_v)$, где

$M_v = \{v_1, v_2, \dots, v_n\}$ – базисное множество,

$p_a\{v_i, v_j\} = \begin{cases} true, j = i + 1 \\ false, j \neq i + 1 \end{cases}$ – отношение следования.

Таким образом, элементы вектора рационально хранить в динамическом массиве (в куче), в котором компоненты вектора имеют шаблонный тип. Вектор определяется двумя параметрами: размером (количество компонент вектора) и стартовым индексом (индекс, начиная с которого можно получить доступ к компонентам вектора).

Пусть заданы два вектора $a = (a_1, a_2, \dots, a_n)$ и $b = (b_1, b_2, \dots, b_n)$. Рассмотрим следующие основные операции над векторами:

- Сравнение векторов $(a == b)$. Вектора считаются равными тогда и только тогда, когда $a_i = b_i$ при всех $i = 1..n$.
- Прибавление скаляра $(a + t)$. Результатом сложения вектора a и скаляра t называется вектор $a' = (a_1 + t, a_2 + t, \dots, a_n + t)$.
- Вычитание скаляра $(a - t)$. Результатом вычитания вектора a и скаляра t называется вектор $a' = (a_1 - t, a_2 - t, \dots, a_n - t)$.
- Умножение на скаляр $(a * t)$. Результатом умножения вектора a на скаляр t называется вектор $a' = (a_1 * t, a_2 * t, \dots, a_n * t)$.
- Сложение векторов $(a + b)$. Результатом сложения векторов a и b называется вектор $c = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$.
- Вычитание векторов (a – b). Результатом вычитания векторов a и b называется вектор $c = (a_1 - b_1, a_2 - b_2, \dots, a_n - b_n)$.
- Скалярное произведение векторов $(a * b)$. Скалярным произведением векторов a и b называется скалярная величина c $= \sum_{i=1}^{n} a_i \times b_i$.

3.1.2 Верхнетреугольные матрицы

Верхнетреугольная матрица – это квадратная матрица, у которой все элементы ниже главной диагонали равны нулю: $a_{ij} = 0$ при $i > j$.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

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

$S2_a = (M2_a, p4_a)$, где

$M2_a = \{v_1, v_2, \dots, v_n\}$ – базисное множество, где v_i есть вектор из i элементов.

$p4_a\{v_i, v_j\} = \begin{cases} true, j = i + 1 \\ false, j \neq i + 1 \end{cases}$ – отношение следования.

Таким образом, элементы верхнетреугольной матрицы рационально хранить в динамическом массиве из массивов.

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

Пусть заданы две матрицы $A = (a_i)$ и $B = (b_i), i = 1..n$, где $a = (a_1, a_2, \dots, a_n)$ и $b = (b_1, b_2, \dots, b_n)$. Рассмотрим следующие основные операции над матрицами:

- Сравнение матриц $(A == B)$. Матрицы считаются равными тогда и только тогда, когда $a_i = b_i$ при всех $i = 1..n$.
- Сложение матриц $(A + B)$. Результатом сложения матриц A и B называется матрица $C = (c_i)$, где $c_i = a_i + b_i$ при всех $i = 1..n$.
- Вычитание матриц $(A - B)$. Результатом вычитания матриц A и B называется матрица $C = (c_i)$, где $c_i = a_i - b_i$ при всех $i = 1..n$.
- Умножение матриц $(A * B)$. Результатом умножения матриц $A = (a_{i,j})$ и $B = (b_{i,j})$ называется матрица $C = (c_{i,j})$, где $c_{ij} = \sum_{k=1}^{n} a_{i,k-startIndex} * b_{k,j-startIndex}$ при всех $i = 1..n; j = 1..n$.

3.2 Описание классов

3.2.1 Класс TVector

Объявление класса:

```
class TVector {
protected:
	int size;
	int startIndex;
	ValueType* pVector;
public:
	TVector(int size = 5, int startIndex = 0);
	TVector(const TVector<ValueType>& v);
	~TVector();
	int GetSize() const;
	int GetStartIndex() const;
	ValueType& operator[](const int i);
	int operator==(const TVector<ValueType>& v) const;
	int operator!=(const TVector<ValueType>& v) const;
	const TVector& operator=(const TVector<ValueType>& v);
	TVector operator+(const ValueType t);
	TVector operator-(const ValueType t);
	TVector operator*(const ValueType t);
	TVector operator+(const TVector<ValueType>& v);
	TVector operator-(const TVector<ValueType>& v);
	double operator*(const TVector<ValueType>& v);
	friend istream& operator>>(istream& in, TVector<ValueType>& v){
		for (int i = 0; i < v.size; i++)
			in >> v.pVector[i];
		return in;
	}
	friend ostream& operator<<(ostream& out, const TVector<ValueType>& v){
		for (int i = 0; i < v.size; i++)
			out << v.pVector[i] << " ";
		return out;
	}
};
```

Поля:

size – количество элементов вектора.

startIndex – индекс первого элемента вектора.

pVector – указатель типа **ValueType** на первый элемент вектора.

Конструкторы:

```
TVector(int size = 5, int startIndex = 0);
```

Назначение: инициализация полей класса **TVector** и выделение памяти под хранение элементов вектора.

Входные данные: **size** – количество элементов вектора, **startIndex** – индекс первого элемента вектора.

Выходные данные: отсутствуют.

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

Назначение: создание копии вектора.

Входные данные: **`TVector<ValueType>& v`** – константная ссылка на вектор.

Выходные данные: отсутствуют.

Деструктор:

```
~TBitField();
```

Назначение: освобождение памяти, занимаемой динамическими полями класса **`TVector`**.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

Методы:

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

Назначение: получение длины вектора.

Входные данные: отсутствуют.

Выходные данные: длины вектора.

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

Назначение: получение индекса первого элемента вектора.

Входные данные: отсутствуют.

Выходные данные: индекс первого элемента вектора.

Операторы:

```
ValueType& operator[](const int i);
```

Назначение: получение элемента вектора под конкретным индексом.

Входные данные: **`i`** – индекс элемента вектора.

Выходные данные: значение элемента вектора под конкретным индексом.

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

Назначение: проверка на равенство векторов.

Входные данные: **`const TVector<ValueType>& v`** – константная ссылка на вектор.

Выходные данные: результат сравнения (1 – векторы равны, 0 – векторы не равны).

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

Назначение: проверка на неравенство векторов.

Входные данные: **`const TVector<ValueType>& v`** – константная ссылка на вектор.

Выходные данные: результат сравнения (1 – векторы не равны, 0 – векторы равны).

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

Назначение: присваивание объекту *this константной ссылки на вектор **`v`**.

Входные данные: **`const TVector<ValueType>& v`** – константная ссылка на вектор.

Выходные данные: ссылка на вектор.

`TVector operator+(const ValueType t);`

Назначение: сложение вектора со скаляром.

Входные данные: **`t`** – скаляр.

Выходные данные: полученный вектор.

`TVector operator-(const ValueType t);`

Назначение: разность вектора со скаляром.

Входные данные: **`t`** – скаляр.

Выходные данные: полученный вектор.

`TVector operator*(const ValueType t);`

Назначение: умножение вектора на скаляр.

Входные данные: **`t`** – скаляр.

Выходные данные: полученный вектор.

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

Назначение: сложение векторов.

Входные данные: `const TVector<ValueType>& v` – константная ссылка на вектор.

Выходные данные: полученный вектор.

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

Назначение: разность векторов.

Входные данные: `const TVector<ValueType>& v` – константная ссылка на вектор.

Выходные данные: полученный вектор.

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

Назначение: скалярное произведение векторов.

Входные данные: `const TVector<ValueType>& v` – константная ссылка на вектор.

Выходные данные: полученное число.

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

Назначение: ввод вектора.

Входные данные: `istream &in` – ссылка на стандартный поток ввода,

`TVector<ValueType>& v` – ссылка на вектор.

Выходные данные: ссылка на поток ввода.

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

Назначение: вывод вектора.

Входные данные: `ostream &out` – ссылка на стандартный поток вывода,

`const TVector<ValueType>& v` – константная ссылка на вектор.

Выходные данные: ссылка на поток вывода.

3.2.2 Класс TMatrix

Объявление класса:

```
class TMatrix : public TVector<TVector<ValueType>> {
public:
	TMatrix(int size = 5);
	TMatrix(const TMatrix<ValueType>& m);
	TMatrix(const TVector<TVector<ValueType>>& v);
	int operator==(const TMatrix<ValueType>& m) const;
	int operator!=(const TMatrix<ValueType>& m) const;
	const TMatrix& operator=(const TMatrix<ValueType>& m);
	TMatrix operator+(const TMatrix<ValueType>& m);
	TMatrix operator-(const TMatrix<ValueType>& m);
	TMatrix operator*(const TMatrix<ValueType>& m);
	friend istream& operator>>(istream& in, TMatrix<ValueType>& m){
		for (int i = 0; i < m.size; i++)
			in >> m.pVector[i];
		return in;
	}
	friend ostream& operator<<(ostream& out, const TMatrix<ValueType>& m){
		for (int i = 0; i < m.size; i++)
		{
			for (int j = 0; j < m.pVector[i].GetStartIndex(); j++) {
				out << "0" << " ";
			}
			out << m.pVector[i] << endl;
		}
		return out;
	}
};
```

Конструкторы:

`TMatrix(int size = 5);`

Назначение: инициализация полей класса **`TVector`** и выделение памяти под хранение элементов вектора.

Входные данные: **`size`** – размер матрицы.

Выходные данные: отсутствуют.

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

Назначение: создание копии матрицы.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: отсутствуют.

`TMatrix(const TVector<TVector<ValueType>>& v);`

Назначение: формирование объекта класса **`TMatrix`** из объекта класса **`TVector`**.

Входные данные: константная ссылка на битовое поле **`bf`**.

Выходные данные: отсутствуют.

Операторы:

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

Назначение: проверка на равенство матриц.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: результат сравнения (1 – матрицы равны, 0 – матрицы не равны).

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

Назначение: проверка на неравенство матриц.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: результат сравнения (1 – матрицы не равны, 0 – матрицы равны).

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

Назначение: присваивание объекту *this константной ссылки на матрицу **`m`**.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: ссылка на матрицу.

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

Назначение: сложение матриц.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: полученная матрица.

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

Назначение: разность матриц.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: полученная матрица.

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

Назначение: умножение матриц.

Входные данные: **`const TMatrix<ValueType>& m`** – константная ссылка на матрицу.

Выходные данные: полученная матрица.

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

Назначение: ввод матрицы.

Входные данные: **`istream &in`** – ссылка на стандартный поток ввода,

`TMatrix<ValueType>& m` – ссылка на матрицу.

Выходные данные: ссылка на поток ввода.

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

Назначение: вывод матрицы.

Входные данные: **`istream &out`** – ссылка на стандартный поток вывода,

`const TMatrix<ValueType>& m` – константная ссылка на матрицу.

Выходные данные: ссылка на поток вывода.

Заключение

В ходе лабораторной работы были изучены основные термины и понятия, связанные с векторами и матрицами, а также наиболее эффективные способы их представления (хранения).

На основе подготовленной теоретической базы, были реализованы классы для представления вектора TVector и верхнетреугольной матрицы TMatrix со всеми необходимыми операциями. Для проверки работоспособности и эффективности реализации перечисленных выше классов были написаны приложения sample_tvector и sample_tmatrix, а также модульные тесты.

Литература

1. Барышева И.В. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017 – 105 с.

2. Вектор [https://blog.skillfactory.ru/glossary/vektor].

3. Матричная алгебра [https://davidprowse.github.io/matlab_course/2023-09-11-octave-matrix-algebra.html].

Приложения

Приложение А. Реализация класса TVector

```
template <typename ValueType>
TVector<ValueType>::TVector(int size, int startIndex) : size(size),
startIndex(startIndex) {
	if (size <= 0)
		throw exception("vector size should be greater than zero");
	if (startIndex < 0)
		throw exception("vector start index should be at least zero");
	pVector = new ValueType[size]();
}

template <typename ValueType>
TVector<ValueType>::TVector(const TVector<ValueType>& v): size(v.size),
startIndex(v.startIndex){
	pVector = new ValueType[size];
	for (int i = 0; i < size; i++)
		pVector[i] = v.pVector[i];
}

template <typename ValueType>
TVector<ValueType>::~TVector() {
	delete[] pVector;
}

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

template <typename ValueType>
int TVector<ValueType>::GetStartIndex() const{
	return startIndex;
}

template <typename ValueType>
ValueType& TVector<ValueType>::operator[](const int i){
	if (i < 0 || i >= size)
		throw exception("out of range");
	return pVector[i];
}
template <typename ValueType>
int TVector<ValueType>::operator==(const TVector<ValueType>& v) const{
	if (size != v.size || startIndex != v.startIndex)
		return 0;
	for (int i = 0; i < size; i++)
		if (pVector[i] != v.pVector[i])
			return 0;
	return 1;
}

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

template <typename ValueType>
const TVector<ValueType>& TVector<ValueType>::operator=(const TVector<ValueType>& v) {
	if (this == &v)
		return *this;
	if (size != v.size || startIndex != v.startIndex) {
		delete[] pVector;
		size = v.size;
		startIndex = v.startIndex;
		pVector = new ValueType[size];
	}
```

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

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator+(const ValueType t) {
	TVector<ValueType> tmp(*this);
	for (int i = 0; i < size; i++)
		tmp[i] += t;
	return tmp;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator-(const ValueType t) {
	TVector<ValueType> tmp(*this);
	for (int i = 0; i < size; i++)
		tmp[i] -= t;
	return tmp;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator*(const ValueType t) {
	TVector<ValueType> tmp(*this);
	for (int i = 0; i < size; i++)
		tmp[i] *= t;
	return tmp;
}
template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator+(const TVector<ValueType>& v) {
	if (size != v.size || startIndex != v.startIndex)
		throw exception("different sizes");
	TVector<ValueType> tmp(*this);
	for (int i = 0; i < size; i++)
		tmp.pVector[i] = tmp.pVector[i] + v.pVector[i];
	return tmp;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator-(const TVector<ValueType>& v) {
	if (size != v.size || startIndex != v.startIndex)
		throw exception("different sizes");
	TVector<ValueType> tmp(*this);
	for (int i = 0; i < size; i++)
		tmp.pVector[i] = tmp.pVector[i] - v.pVector[i];
	return tmp;
}

template <typename ValueType>
double TVector<ValueType>::operator*(const TVector<ValueType>& v) {
	if (size != v.size || startIndex != v.startIndex)
		throw exception("different sizes");
	double sum = 0.0;
	for (int i = 0; i < size; i++)
		sum += pVector[i] * v.pVector[i];
	return sum;
}
```

Приложение Б. Реализация класса TMatrix

```
template <typename ValueType>
TMatrix<ValueType>::TMatrix(int size): TVector<TVector<ValueType>>(size){
	for (int i = 0; i < size; ++i) {
		pVector[i] = TVector<ValueType>(size - i, i);
	}
}

template <typename ValueType>
```

```
TMatrix<ValueType>::TMatrix(const TMatrix<ValueType>& m):
TVector<TVector<ValueType>>(m){}

template <typename ValueType>
TMatrix<ValueType>::TMatrix(const TVector<TVector<ValueType>>& v):
TVector<TVector<ValueType>>(v){}

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

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

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

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator+(const TMatrix<ValueType>& m)
{
	if (size != m.size) {
		throw exception("different sizes");
	}
	TMatrix tmp(*this);
	for (int i = 0; i < size; i++)
		tmp.pVector[i] = tmp.pVector[i] + m.pVector[i];
	return tmp;
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator-(const TMatrix<ValueType>& m)
{
	if (size != m.size) {
		throw exception("different sizes");
	}
	TMatrix tmp(*this);
	for (int i = 0; i < size; i++)
		tmp.pVector[i] = tmp.pVector[i] - m.pVector[i];
	return tmp;
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator*(const TMatrix<ValueType>& m) {
	if (size != m.size)
		throw exception("different sizes");
	TMatrix<ValueType> tmp(size);
	for (int i = 0; i < size; i++)
		for (int j = i; j < size; j++) {
			tmp[i][j - tmp.pVector[i].GetStartIndex()] = 0;
			for (int k = i; k <= j; k++)
				tmp.pVector[i][j - tmp.pVector[i].GetStartIndex()] += this-
>pVector[i][k - this->pVector[i].GetStartIndex()] * m.pVector[k][j -
m.pVector[k].GetStartIndex()];
		}
	return tmp;
}
```