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 3](#_Toc153298816)

[1 Постановка задачи 4](#_Toc153298817)

[2 Руководство пользователя 5](#_Toc153298818)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc153298819)

[2.2 Приложение для демонстрации работы верхнетреугольных матриц 7](#_Toc153298820)

[3 Руководство программиста 9](#_Toc153298821)

[3.1 Использованные алгоритмы 9](#_Toc153298822)

[3.1.1 Векторы 9](#_Toc153298823)

[3.1.2 Верхнетреугольные матрицы 10](#_Toc153298824)

[3.2 Описание классов 11](#_Toc153298825)

[3.2.1 Класс TVector 11](#_Toc153298826)

[3.2.2 Класс TMatrix 15](#_Toc153298827)

[Заключение 17](#_Toc153298828)

[Литература 19](#_Toc153298829)

[Приложения 20](#_Toc153298830)

[Приложение А. Реализация класса TVector 20](#_Toc153298831)

[Приложение Б. Реализация класса TMatrix 21](#_Toc153298832)

# 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.

# Постановка задачи

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

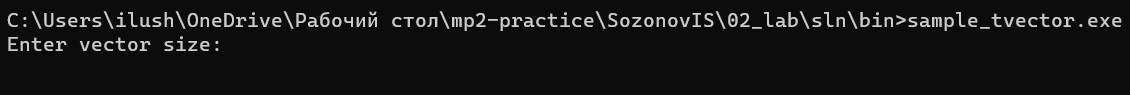
Задачи:

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.

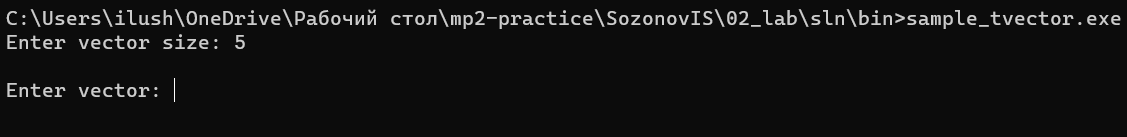
# Руководство пользователя

## Приложение для демонстрации работы векторов

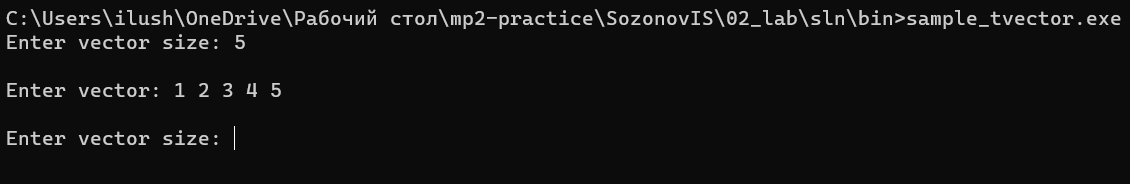
1. Запустить sample\_tvector.exe. В результате появится окно для ввода длины вектора a (рис. 1).



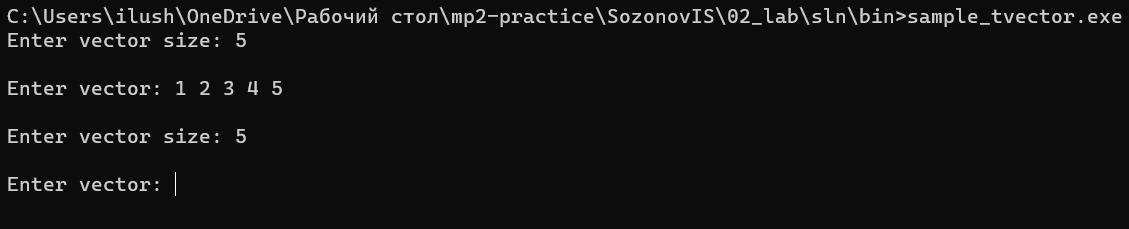
1. Основное окно приложения
2. Ввести длину вектора. В результате появится окно для ввода вектора a (Рис. 2).



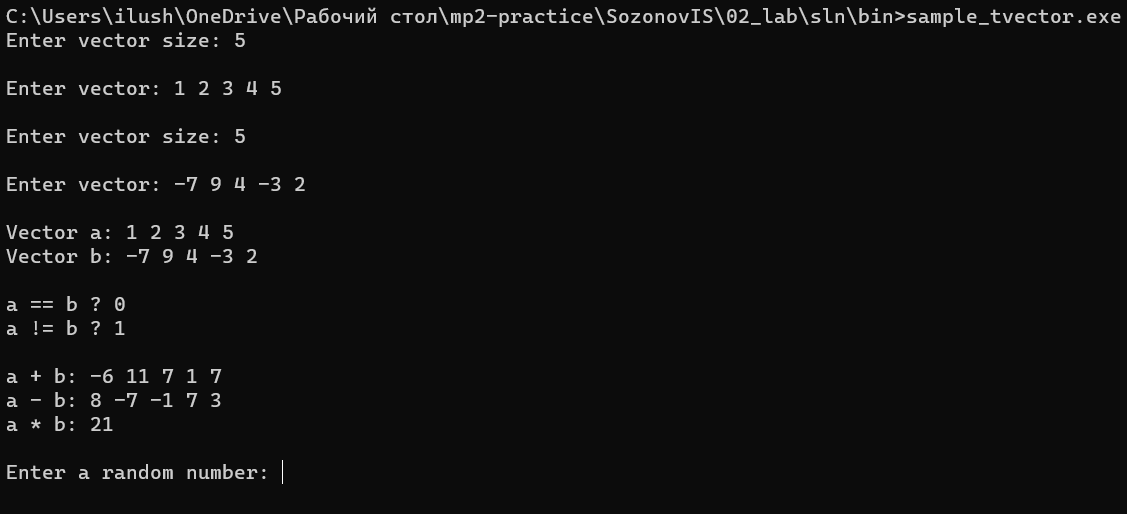
1. Ввод длины вектора a
2. Ввести вектор a. В результате появится окно для ввода длины вектора b (Рис. 3).



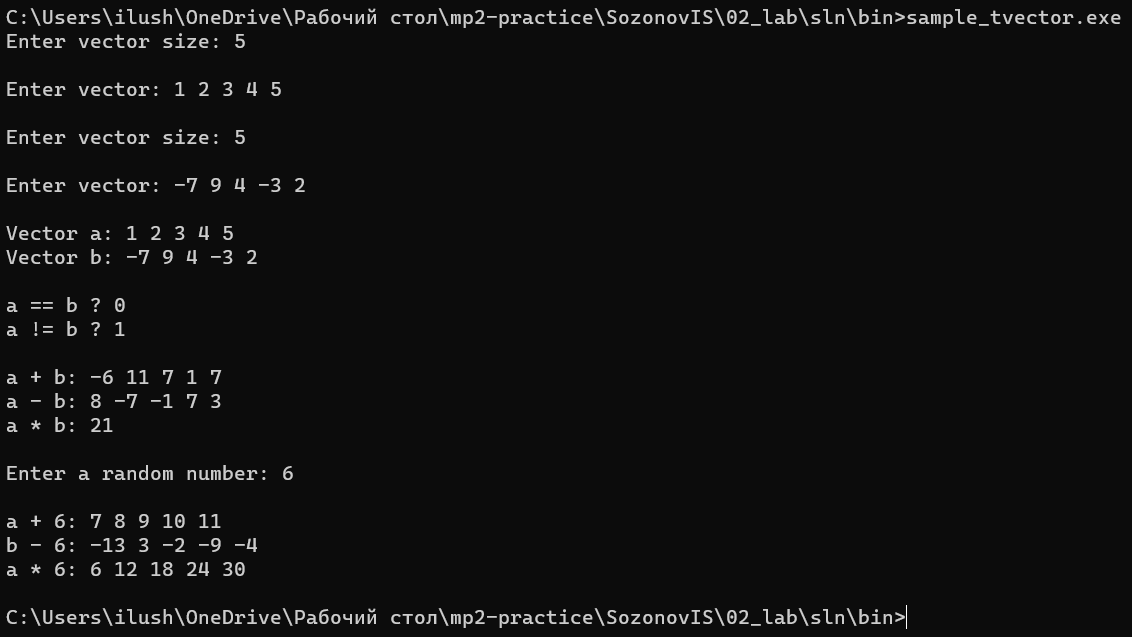
1. Ввод вектора a
2. Ввести длину вектора b. В результате появится окно для ввода вектора b (Рис. 4).



1. Ввод длины вектора b
2. 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).



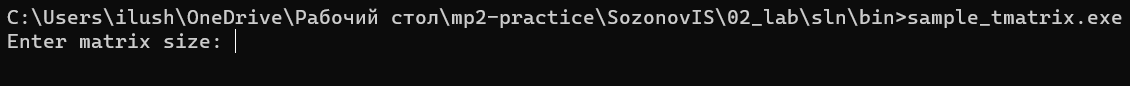
1. Ввод вектора b
2. Ввести случайное число. В результате будут выведены результаты сложения вектора a с введенным числом, разности вектора b с введенным числом и умножения вектора a на введенное число (Рис. 6).



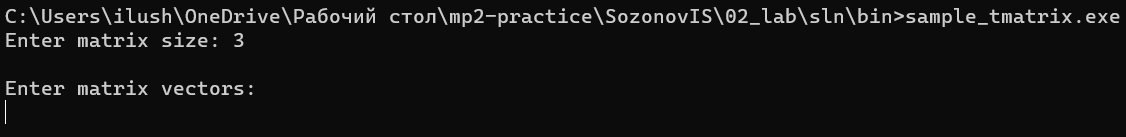
1. Ввод случайного числа

## Приложение для демонстрации работы верхнетреугольных матриц

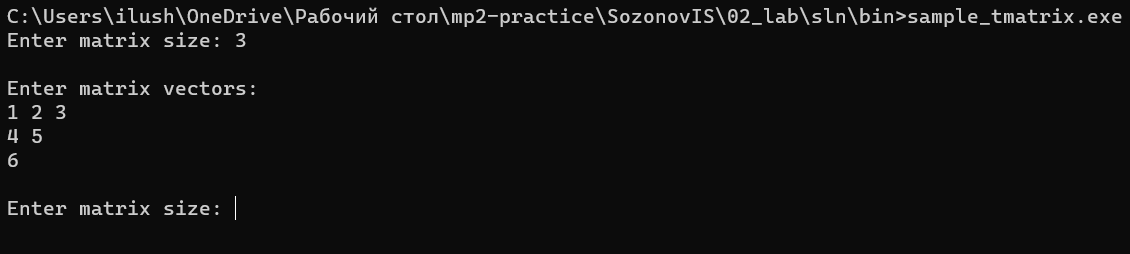
1. Запустить sample\_tmatrix.exe. В результате появится окно для ввода размера верхнетреугольной матрицы A (Рис. 7).



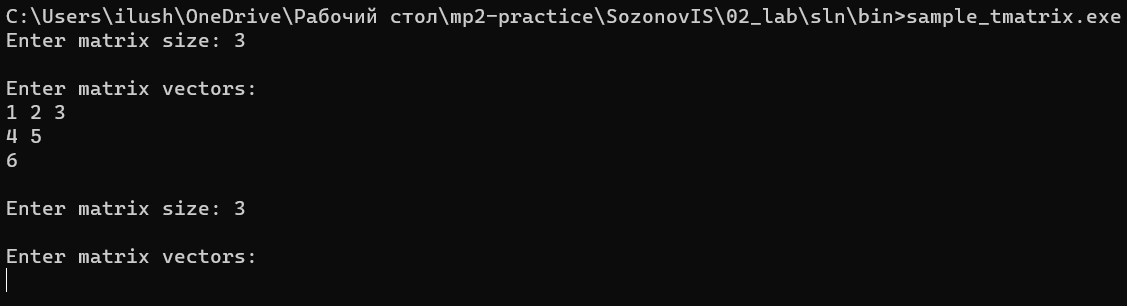
1. Основное окно приложения
2. Ввести размер матрицы A. В результате появится окно для ввода векторов матрицы A (**Ошибка! Источник ссылки не найден.**).



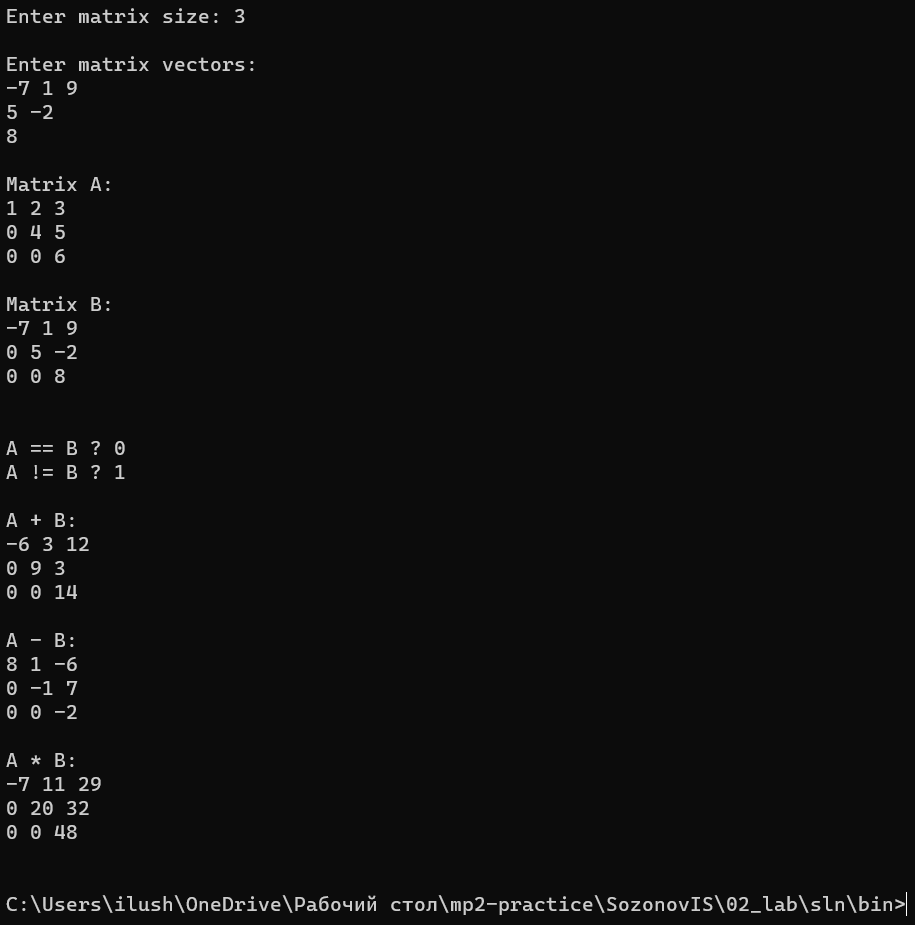
1. Ввод размера матрицы A
2. Ввести векторы матрицы A. В результате появится окно для ввода размера верхнетреугольной матрицы B (**Ошибка! Источник ссылки не найден.**).



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



1. Ввод размера матрицы B
2. 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.**).



1. Ввод мощности второго множества

# Руководство программиста

## Использованные алгоритмы

### Векторы

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

Структура данных вектора есть

, где

– базисное множество,

– 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 и . Рассмотрим следующие основные операции над векторами:

* Сравнение векторов . Вектора считаются равными тогда и только тогда, когда при всех .
* Прибавление скаляра . Результатом сложения вектора *a* и скаляра *t* называется вектор .
* Вычитание скаляра . Результатом вычитания вектора *a* и скаляра *t* называется вектор .
* Умножение на скаляр . Результатом умножения вектора *a* на скаляр *t* называется вектор .
* Сложение векторов . Результатом сложения векторов *a* и *b* называется вектор .
* Вычитание векторов (a – b). Результатом вычитания векторов *a* и *b* называется вектор .
* Скалярное произведение векторов . Скалярным произведением векторов *a* и *b* называется скалярная величина .

### Верхнетреугольные матрицы

Верхнетреугольная матрица – это квадратная матрица, у которой все элементы ниже главной диагонали равны нулю: при .

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

, где

– базисное множество, где есть вектор из *i* элементов.

– 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 и ,, где и . Рассмотрим следующие основные операции над матрицами:

* Сравнение матриц . Матрицы считаются равными тогда и только тогда, когда при всех .
* Сложение матриц . Результатом сложения матриц *A* и *B* называется матрица, где при всех .
* Вычитание матриц . Результатом вычитания матриц *A* и *B* называется матрица , где при всех .
* Умножение матриц . Результатом умножения матриц и называется матрица , где при всех ; .

## Описание классов

### 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.

### 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;

}