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:

**«Матрица на основе вектора»**

**Выполнил(а):** студент(ка) группы ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фатехов К.Г./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2023

Содержание

[Введение 3](#_Toc149043756)

[1 Постановка задачи 4](#_Toc149043757)

[2 Руководство пользователя 5](#_Toc149043758)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc149043759)

[2.2 Приложение для демонстрации работы матриц 6](#_Toc149043760)

[3 Руководство программиста 9](#_Toc149043762)

[3.1 Описание алгоритмов 9](#_Toc149043763)

[3.1.1 Векторы 9](#_Toc149043764)

[3.1.2 Матрицы 11](#_Toc149043765)

[3.2 Описание программной реализации 14](#_Toc149043767)

[3.2.1 Описание класса TVector 14](#_Toc149043768)

[3.2.2 Описание класса TMatrix 17](#_Toc149043769)

[Заключение 22](#_Toc149043770)

[Литература 23](#_Toc149043771)

[Приложения 24](#_Toc149043772)

[Приложение А. Реализация класса TVector 24](#_Toc149043773)

[Приложение Б. Реализация класса TMatrix 26](#_Toc149043774)

# Введение

Программа "Матрица на основе шаблонных векторов" является мощным инструментом для работы с математическими матрицами. Она основана на использовании шаблонных векторов, что позволяет достичь гибкости и универсальности в работе с различными типами данных.

Данная программа предоставляет возможность создания и манипулирования матрицами произвольного размера. Она позволяет выполнить такие операции, как создание пустой матрицы, заполнение ее элементов, выполнение арифметических операций с матрицами (сложение, вычитание, умножение).

Преимущество программы "Матрица на основе шаблонных векторов" состоит в том, что она способна работать с различными числовыми типами данных (целочисленными, вещественными и т. д.), а также с пользовательскими типами данных, благодаря применению шаблонных векторов. Это позволяет легко вписаться в широкий спектр задач, связанных с матричными вычислениями.

Кроме того, программа обладает простым и интуитивно понятным интерфейсом, что делает ее использование легким даже для начинающих пользователей. Она также имеет высокую производительность и оптимизирована для работы с большими объемами данных.

# Постановка задачи

Цель: Разработать матрицу на основе шаблонного вектора на языке программирования C++. Матрица включает в себя 2 класса «TVector» и «TMatrix» В этих классах нужно сделать реализацию основных операций с векторами и матрицами.

2. Задачи данной лабораторной работы:

- Разработка класса TVector и реализация основных операций для работы с ними.

- Разработка класса TMatrix и реализация основных операций для работы с ними.

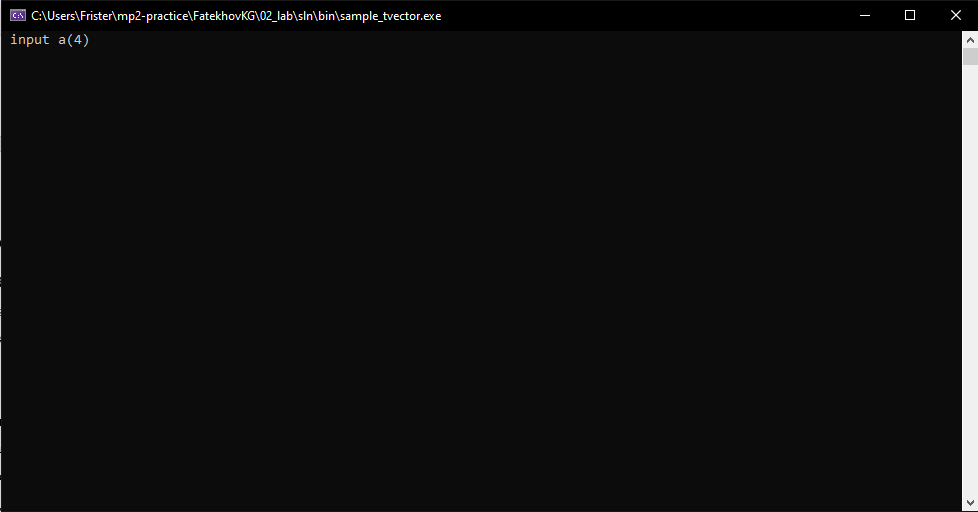
- Проверка и демонстрация работы разработанных классов с помощью приложений для работы с матрицами и векторами.

- Написание отчета о выполненной лабораторной работе.

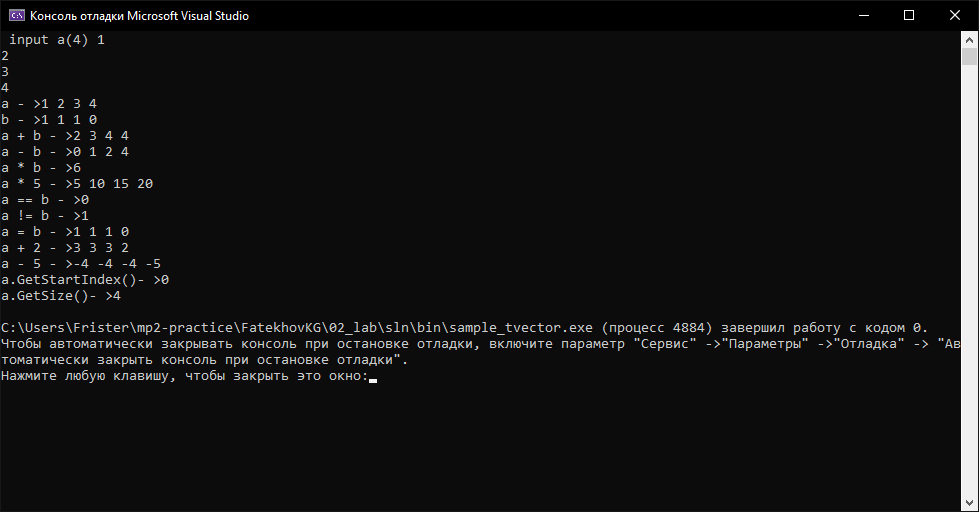
# Руководство пользователя

## Приложение для демонстрации работы векторов

1. Запустим приложение с названием sample\_tvector.exe. В результате появится следующее окно



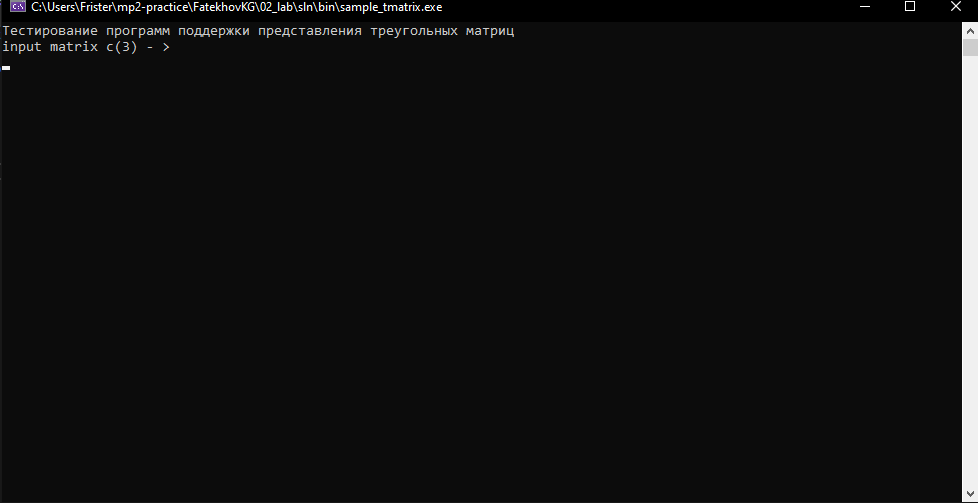
1. Основное окно программы
2. Это окно показывает работу основных функций работы с векторами (сравнение на равенство, присваивание, длина, сложение, скалярное умножение). Предлагается ввести вектор длины 4. В результате будет выведено (рис. 2).
3. Окно демонстрирует все операции связанные с векторами, описанные в данной программе.



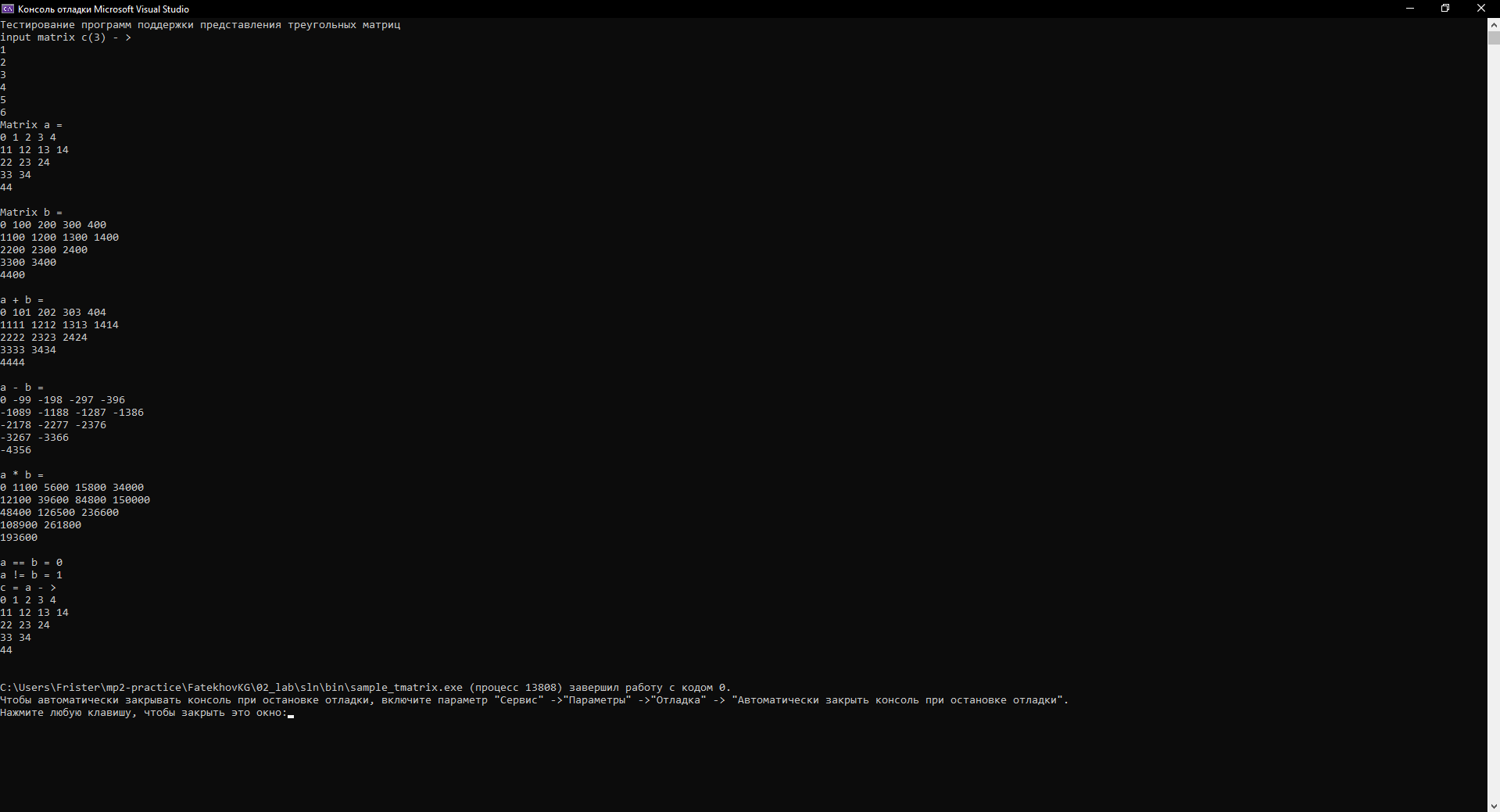
1. Основное окно программы

## Приложение для демонстрации работы матриц

1. Запустим приложение с названием sample\_tmatrix.exe. В результате появится следующее окно



1. Основное окно программы
2. Предлагается ввести матрицу размера 3. Будет выведено следующее (рис. 4). Это окно показывает работу основных функций работы с матрицами (умножение и сложение матриц).



1. Основное окно программы

# Руководство программиста

## Описание алгоритмов

### Векторы

Начало работы указано в пункте [2.1](#_Приложение_для_демонстрации_1)

Описание методов и полей класса указано в пункте [3.2.1](#_Описание_класса_TBitField)

Векторы в программировании представляют собой упорядоченные наборы элементов одного типа, которые могут изменяться в размере. Они представляют собой динамические структуры данных и часто используются для хранения и манипуляции коллекциями элементов. Они обладают следующими преимуществами:  
1. Динамический размер: Векторы позволяют добавлять и удалять элементы, что позволяет динамически изменять их размер, в отличие от статических массивов.

2. Быстрый доступ к элементам: Векторы обеспечивают быстрый доступ к элементам за счет использования индексов. Это позволяет быстро получать доступ к любому элементу вектора.

3. Поддержка итераций: Векторы позволяют легко выполнять итерации по элементам с помощью циклов, что делает их удобными для обработки больших объемов данных.

4. Универсальность: Векторы поддерживают элементы любого типа данных, что делает их полезными для хранения и обработки различных структур данных.

Операция GetSize():

Операция возвращает размер вектора.

Пример:

1 2 3

3

Операция GetStartIndex():

Операция возвращает индекс первого элемента вектора.

Пример:

Для v(3,1)

1

**Операция индексирования**:

Операция возвращает элемент вектора стоящий в указанной позиции.

Пример:

V1 = (1 2 3)

V1[2] = 3

**Операция ввода вектора из консоли:**

Операция позволяет ввести вектор с консоли. Вводим элементы вектора, пока не достигнем установленной длины.

Пример:

0 0 0 0 – вектор длины 4

После ввода:

2 3 3 3

**Операция вывода вектора в консоль:**

Операция вывода позволяет вывести вектор в консоль

**Операция сравнения:**

Операция возвращает 0, когда вектора не равны и 1 в противном случае

Пример:

V1 = (1 1 1)

V2 = (0 0 0)

C = (V1 == V2) = 0

**Скалярные операции:**

Операции возвращают экземпляр вектора, все элементы которого изменились на указанное число.

Пример:

1 2 3

+ 3

4 5 6

**Операция прибавление вектора:**

Операция возвращает экземпляр класса, результат сложения двух векторов.

Пример:

V1 = (1 1 1)

V2 = (2 2 2)

V3 = (3 3 3)

**Операция умножения векторов:**

Операция возвращает число, результат умножения двух векторов скалярно.

Пример:

V1 = (1 1 1)

V2 = (2 2 2)

C = V1 \* V2 = 6

**Операция вычитания векторов:**

Операция возвращает экземпляр класса, результат вычитания двух векторов.

Пример:

V1 = (2 2 2)

V2 = (1 1 1)

V3 = V1 – V2 = (1 1 1)

### Матрицы

Начало работы указано в пункте [2.2](#_Приложение_для_демонстрации)

Описание методов и полей класса указано в пункте [3.2.2](#_Описание_класса_TSet)

Матрица представляет собой двумерный массив, состоящий из элементов одного типа данных. Она представляет собой таблицу, где каждый элемент находится в определенной строке и столбце. Матрицы используются для хранения и обработки данных, таких как числа, символы, строки и т. д.

Матрица реализована на основе шаблонного класса вектор.

Матрица поддерживает наследуемые поля вектора, а также его методы.

**Операция сложения матриц:**

Операция возвращает экземпляр класса, результат сложения матриц.

Пример:

A = 0 1 2 3 4

11 12 13 14

22 23 24

33 34

44

B = 0 100 200 300 400

1100 1200 1300 1400

2200 2300 2400

3300 3400

4400

Результат сложения матриц A и B:

A+B = 0 101 202 303 404

1111 1212 1313 1414

2222 2323 2424

3333 3434

4444

**Операция вычитания матриц:**

Операция возвращает экземпляр класса, результат вычитания матриц.

A =

1 1 1

1 1

1

B =

1 1 1

1 1

0

A – B =

0 0 0

0 0

1

**Операция умножения матриц:**

A =

0 1 2 3 4

11 12 13 14

22 23 24

33 34

44

B =

0 100 200 300 400

1100 1200 1300 1400

2200 2300 2400

3300 3400

4400

A \* B =

0 1100 5600 15800 34000

12100 39600 84800 150000

48400 126500 236600

108900 261800

193600

**Операция индексации:**

Операция нужна для получения элемента матрицы, которым в свою очередь является вектор – строка.

Пример:

A =

0 1 2 3 4

11 12 13 14

22 23 24

33 34

44

A[0] = 0 1 2 3 4

A[0][0] = 0

**Операция сравнения на равенство:**

Операция возвращает 1, если матрицы равны поэлементно, а также равны размеры и стартовые индексы, 0 в противном случае.

Пример:

A =

1 1 1

1 1

1

B =

1 1 1

1 1

1

A = B = 1;

## Описание программной реализации

### Описание класса TVector

template <class ValType>

class TVector

{

protected:

ValType\* pVector;

int Size;

int StartIndex;

public:

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

TVector(const TVector& v);

~TVector();

int GetSize() { return Size; }

int GetStartIndex() { return StartIndex; }

ValType& operator[](int pos);

//ValType& operator[](int pos) const; // доступ

bool operator==(const TVector& v) const; // сравнение

bool operator!=(const TVector& v) const; // сравнение

TVector& operator=(const TVector& v); // присваивание

// скалярные операции

TVector operator+(const ValType& val);

TVector operator-(const ValType& val);

TVector operator\*(const ValType& val);

// векторные операции

TVector operator+(const TVector& v);

TVector operator-(const TVector& v);

ValType operator\*(const TVector& v);

// ввод-вывод

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

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

};

Назначение: представление вектора.

Поля:

\*pVector – память для элементов вектора

Size – количество элементов вектора

StartIndex – индекс первого элемента вектора

Методы:

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

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры:

s – длинна вектора (по умолчанию 10).

si – стартовый индекс (по умолчанию 0).

Выходные параметры: отсутствуют.

TVector(const TVector<ValType>& v);

Назначение: конструктор копирования.

Входные параметры:

v – экземпляр класса, на основе которого создаем новый объект.

Выходные параметры: отсутствуют.

~TVector();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

int GetSize();

Назначение: получение размера вектора.

Входные параметры: отсутствуют.

Выходные параметры: размер вектора ( количество элементов).

int GetStartIndex();

Назначение: получение стартового индекса.

Входные параметры: отсутствуют.

Выходные параметры: стартовый индекс.

ValType& operator[](int pos);

Назначение: перегрузка оператора индексации.

Входные параметры:

pos – позиция (индекс) элемента.

Выходные параметры: элемент, который находится на pos позиции.

bool operator==(const TVector<ValType> &v) const;

Назначение: оператор сравнения.

Входные параметры:

v – экземпляр класса, с которым сравниваем.

Выходные параметры:

true (1), если они равны, иначе false(0).

bool operator!=(const TVector<ValType> &v) const;

Назначение: оператор сравнения.

Входные параметры:

v – экземпляр класса, с которым сравниваем.

Выходные параметры:

true (1), если они не равны, иначе false(0).

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

Назначение: оператор присваивания.

Входные параметры:

v – экземпляр класса, который присваиваем.

Выходные параметры:

Ссылка на присвоенный экземпляр класса.

TVector<ValType> operator+(const ValType &val);

Назначение: оператор суммирования вектора и значения.

Входные параметры:

Val – элемент, с которым суммируем.

Выходные параметры:

Экземпляр класса, элементы которого на val больше.

TVector<ValType> operator-(const ValType &val);

Назначение: оператор вычитания вектора и значения.

Входные параметры:

Val – элемент, который вычитаем.

Выходные параметры:

Экземпляр класса, элементы которого на val меньше.

TVector<ValType> operator\*(const ValType &val);

Назначение: оператор умножения вектора на значение.

Входные параметры:

Val – элемент, на который умножаем вектор.

Выходные параметры:

Экземпляр класса, элементы которого в val раз больше.

TVector<ValType> operator+(const TVector<ValType> &v);

Назначение: оператор суммирования векторов.

Входные параметры:

V – вектор, который суммируем.

Выходные параметры:

Экземпляр класса, равный сумме двух векторов.

TVector<ValType> operator-(const TVector<ValType> &v);

Назначение: оператор вычитания векторов.

Входные параметры:

V – вектор, который вычитаем.

Выходные параметры:

Экземпляр класса, равный разности двух векторов.

TVector<ValType> operator\*(const TVector<ValType> &v);

Назначение: оператор умножения векторов.

Входные параметры:

V – вектор, на который умножаем.

Выходные параметры:

Значение, равное скалярному произведению двух векторов.

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

Назначение: оператор ввода вектора.

Входные параметры:

in – ссылка на буфер, из которого вводим вектор.

V – ссылка на вектор, который вводим.

Выходные данные:

in – ссылка буфер.

friend ostream& operator<<(ostream &out,TVector<ValType> &v);

Назначение: оператор вывода вектора

Входные параметры:

in – ссылка на буфер, из которого выводим вектор.

v – ссылка на вектор, который выводим.

Выходные данные:

in – ссылка буфер.

### Описание класса TMatrix

template <class ValType>

class TMatrix : public TVector<TVector<ValType> >

{

public:

TMatrix(int s = 10);

TMatrix(const TMatrix& mt); // копирование

TMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа

bool operator==(const TMatrix& mt) const; // сравнение

bool operator!=(const TMatrix& mt) const; // сравнение

const TMatrix& operator= (const TMatrix& mt); // присваивание

TMatrix operator+ (const TMatrix& mt); // сложение

TMatrix operator- (const TMatrix& mt); // вычитание

TMatrix operator\* (const TMatrix& mt); // умножение

// ввод / вывод

friend istream& operator>>(istream& in, TMatrix& mt);

friend ostream& operator<<(ostream& out, const TMatrix& mt);

};

Назначение: представление матрицы.

Поля:

StartIndex – индекс первого необходимого элемента.

\*pVector – память для представления элементов матрицы.

Size – размерность матрицы.

Методы:

TMatrix(int s = 10);

Назначение: конструктор по умолчанию и конструктор с параметрами

Входные параметры:

s – длина вектора (по умолчанию 10).

Выходные параметры: отсутствуют.

TMatrix(const TMatrix &mt);

Назначение: конструктор копирования

Входные параметры:

mt – экземпляр класса, на основе которого создаем новый объект.

Выходные параметры: отсутствуют.

TMatrix(const TVector<TVector<ValType> > &mt);

Назначение: Конструктор преобразования типов.

Входные параметры:

mt – ссылка на TVector<TVector<ValType>> - на объект, который преобразуем.

Выходные данные: отсутствуют.

bool operator==(const TMatrix<ValType>&mt) const;

Назначение: оператор сравнения.

Входные параметры:

mt – экземпляр класса, с которым сравниваем.

Выходные параметры:

true (1), если они равны, иначе false(0).

bool operator!=(const TMatrix<ValType>&mt) const;

Назначение: оператор сравнения.

Входные параметры:

mt – экземпляр класса, с которым сравниваем.

Выходные параметры:

true (1), если они не равны, иначе false(0).

const TMatrix& operator=(const TMatrix<ValType> &mt);

Назначение: оператор присваивания.

Входные параметры:

mt – экземпляр класса, который присваиваем.

Выходные параметры:

Ссылка на присвоенный экземпляр класса.

TMatrix operator+(const TMatrix<ValType> &mt);

Назначение: оператор суммирования матриц.

Входные параметры:

mt – ссылка на матрицу, которую суммируем.

Выходные параметры:

Экземпляр класса, равный сумме двух матриц.

TMatrix operator-(const TMatrix<ValType> &mt);

Назначение: оператор вычитания матриц.

Входные параметры:

mt – ссылка на матрицу, которую вычитаем.

Выходные параметры:

Экземпляр класса, равный разности двух матриц.

TMatrix operator\*(const TMatrix<ValType> &mt);

Назначение: оператор умножения матриц.

Входные параметры:

mt – ссылка на матрицу, которую умножаем.

Выходные параметры:

Экземпляр класса, равный произведению двух матриц.

friend istream& operator>>(istream &in, TMatrix<ValType>&mt);

Назначение: оператор ввода матрицы.

Входные параметры:

in – ссылка на буфер, из которого вводим матрицу.

v – ссылка на матрицу, которую вводим.

Выходные данные:

in – ссылка буфер.

friend ostream& operator<<(ostream &out, TMatrix<ValType>&mt);

Назначение: оператор вывода матрицы.

Входные параметры:

out – ссылка на буфер, из которого выводим матрицу.

v – ссылка на матрицу, который выводим.

Выходные данные:

our – ссылка буфер.

# Заключение

В ходе выполнения лабораторной работы смогли лучше изучить шаблоны на языке C++. С помощью шаблонов удалось создать вектор, который может использоваться с разными типами данных. Класс вектор содержит основные операции, такие как сложение векторов, вычитание векторов, добавление и удаление элементов, доступ к элементу по индексу и т.д. Также на основе шаблонного класса вектора, удалось разработать шаблонный класс матриц, с помощью которого удалось реализовать верхнетреугольные матрицы. Класс матриц поддерживает операции сложения матриц, умножения матрицы на матрицу и другие.

.

# Литература

1. Лекции Сысоева А.В.
2. Chat GPT

# Приложения

## Приложение А. Реализация класса TVector

template <class ValType>

TVector<ValType>::TVector(int s, int si)

{

if (s < 0 || s > MAX\_VECTOR\_SIZE) throw "invalid size";

Size = s;

if (si < 0 ) throw "StartIndex cant be negative";

StartIndex = si;

pVector = new ValType[Size];

for (int i = 0; i < Size; i++) {

pVector[i] = 0;

}

} /\*-------------------------------------------------------------------------\*/

template <class ValType> //конструктор копирования

TVector<ValType>::TVector(const TVector& v)

{

Size = v.Size;

StartIndex = v.StartIndex;

pVector = new ValType[Size];

for (int i = 0; i < Size; i++) pVector[i] = v.pVector[i];

} /\*-------------------------------------------------------------------------\*/

template <class ValType>

TVector<ValType>::~TVector()

{

delete[] pVector;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // доступ

ValType& TVector<ValType>::operator[](const int pos)

{

if (pos < 0 || pos >= Size + StartIndex) throw "invalid index";

if (pos < StartIndex) throw "invalid ind";

return pVector[pos - StartIndex];

} /\*-------------------------------------------------------------------------\*/

//template <class ValType> // доступ

//ValType& TVector<ValType>::operator[](int pos) const

//{

// return pVector[pos];

//}/\*-------------------------------------------------------------------------\*/

template <class ValType> // сравнение

bool TVector<ValType>::operator==(const TVector<ValType>& v) const

{

if (Size != v.Size) return false;

if (StartIndex != v.StartIndex) return false;

for (int i = 0; i < Size; i++) {

if (pVector[i] != v.pVector[i]) return false;

}

return true;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // сравнение

bool TVector<ValType>::operator!=(const TVector& v) const

{

return !(\*this == v);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // присваивание

TVector<ValType>& TVector<ValType>::operator=(const TVector& v)

{

if (\*this == v) return \*this;

delete[] pVector;

Size = v.Size;

pVector = new ValType[Size];

StartIndex = v.StartIndex;

for (int i = 0; i < Size; i++) {

pVector[i] = v.pVector[i];

}

return \*this;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // прибавить скаляр

TVector<ValType> TVector<ValType>::operator+(const ValType& val)

{

TVector<ValType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp.pVector[i] = pVector[i] + val;

return tmp;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // вычесть скаляр

TVector<ValType> TVector<ValType>::operator-(const ValType& val)

{

TVector<ValType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp.pVector[i] = pVector[i] - val;

return tmp;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // умножить на скаляр

TVector<ValType> TVector<ValType>::operator\*(const ValType& val)

{

TVector<ValType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp.pVector[i] = pVector[i] \* val;

return tmp;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // сложение

TVector<ValType> TVector<ValType>::operator+(const TVector<ValType>& v)

{

if (Size != v.Size) throw "Vector sizes have to be equal";

if (StartIndex != v.StartIndex) throw "Vector start indexes have to be equal";

TVector<ValType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp.pVector[i] = pVector[i] + v.pVector[i];

return tmp;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // вычитание

TVector<ValType> TVector<ValType>::operator-(const TVector<ValType>& v)

{

if (Size != v.Size) throw "Vector sizes have to be equal";

if (StartIndex != v.StartIndex) throw "Vector start indexes have to be equal";

TVector<ValType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp.pVector[i] = pVector[i] - v.pVector[i];

return tmp;

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // скалярное произведение

ValType TVector<ValType>::operator\*(const TVector<ValType>& v)

{

if (Size != v.Size) throw "Vector sizes have to be equal";

if (StartIndex != v.StartIndex) throw "Vector start indexes have to be equal";

ValType tmp = ValType();

for (int i = 0; i < Size; i++) {

tmp += pVector[i] \* v.pVector[i];

}

return tmp;

} /\*-------------------------------------------------------------------------\*/}

## Приложение Б. Реализация класса TSet

template <class ValType>

TMatrix<ValType>::TMatrix(int s) : TVector<TVector<ValType> >(s)

{

if (s < 0) throw "Size cant be negative";

if (s > MAX\_MATRIX\_SIZE) throw "Size is too large";

for (int i = 0; i < s; i++) {

pVector[i] = TVector<ValType>(s - i, i);

}

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // конструктор копирования

TMatrix<ValType>::TMatrix(const TMatrix<ValType>& mt) :

TVector<TVector<ValType> >(mt) {}

template <class ValType> // конструктор преобразования типа

TMatrix<ValType>::TMatrix(const TVector<TVector<ValType> >& mt) :

TVector<TVector<ValType> >(mt) {}

template <class ValType> // сравнение

bool TMatrix<ValType>::operator==(const TMatrix<ValType>& mt) const

{

return TVector<TVector<ValType>>::operator==(mt);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // сравнение

bool TMatrix<ValType>::operator!=(const TMatrix<ValType>& mt) const

{

return !(\*this == mt);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // присваивание

const TMatrix<ValType>& TMatrix<ValType>::operator=(const TMatrix<ValType>& mt)

{

return TVector <TVector<ValType>>::operator=(mt);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // сложение

TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType>& mt)

{

if (Size != mt.Size) throw "Sizes isnt equal";

return TVector<TVector<ValType>>::operator+(mt);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // вычитание

TMatrix<ValType> TMatrix<ValType>::operator-(const TMatrix<ValType>& mt)

{

if (Size != mt.Size) throw "Sizes isnt equal";

return TVector<TVector<ValType>>::operator-(mt);

} /\*-------------------------------------------------------------------------\*/

template <class ValType> // умножение

TMatrix<ValType> TMatrix<ValType>::operator\*(const TMatrix& mt)

{

if (Size != mt.Size) throw "Not equal sizes";

if (StartIndex != mt.StartIndex) throw "Not equal index";

TMatrix<ValType> tmp(mt), res(Size);

for (int i = 0; i < Size; i++)

{

for (int j = i; j < Size; j++)

{

for (int k = i; k <= j; k++)

{

res[i][j] += (\*this)[i][k] \* tmp[k][j];

}

}

}

return res;

}

#endif