

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Национальный исследовательский  
Нижегородский государственный университет им. Н.И. Лобачевского»  
(ННГУ)

Институт информационных технологий, математики и механики

## ЛАБОРАТОРНАЯ РАБОТА

на тему:  
«Матрица на основе вектора»

Выполнил(а): студент(ка) группы \_\_\_\_  
\_\_\_\_\_/ Фатехов К.Г./  
Подпись

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

Нижний Новгород  
2023

## Содержание

|  |  |
|--|--|
| Введение.....  | 3                                      |
| 1 Постановка задачи.....                             | 4                                      |
| 2 Руководство пользователя .....                     | 5                                      |
| 2.1 Приложение для демонстрации работы векторов..... | 5                                      |
| 2.2 Приложение для демонстрации работы матриц .....  | 6                                      |
| 3 Руководство программиста.....                      | 8                                      |
| 3.1 Описание алгоритмов.....                         | 8                                      |
| 3.1.1 Векторы.....                                   | <b>Ошибка! Закладка не определена.</b> |
| 3.1.2 Матрицы.....                                   | 10                                     |
| 3.2 Описание программной реализации.....             | 14                                     |
| 3.2.1 Описание класса TBitField .....                | 14                                     |
| 3.2.2 Описание класса TSet .....                     | 18                                     |
| Заключение .....                                     | 23                                     |
| Литература .....                                     | 24                                     |
| Приложения .....                                     | 25                                     |
| Приложение А. Реализация класса TBitField.....       | 25                                     |
| Приложение Б. Реализация класса TSet.....            | 25                                     |

## 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.

# 1 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.

## 2 Руководство пользователя

### 2.1 Приложение для демонстрации работы векторов

1. Запустим приложение с названием `sample_tvector.exe`. В результате появится следующее окно

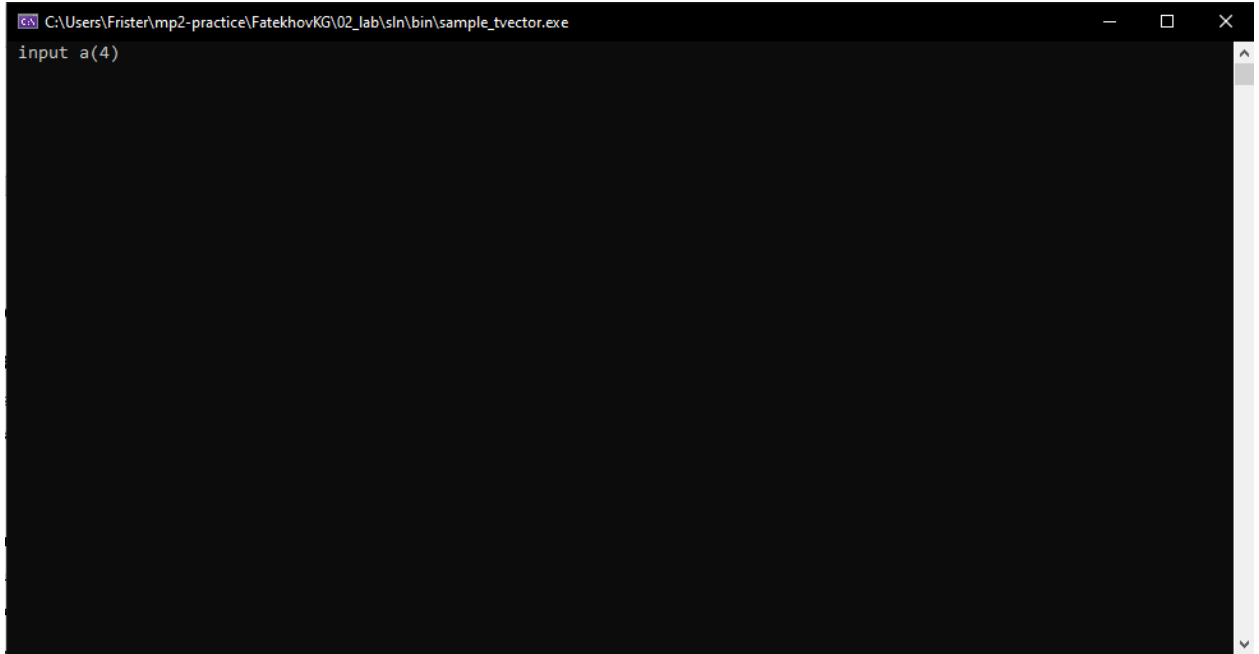
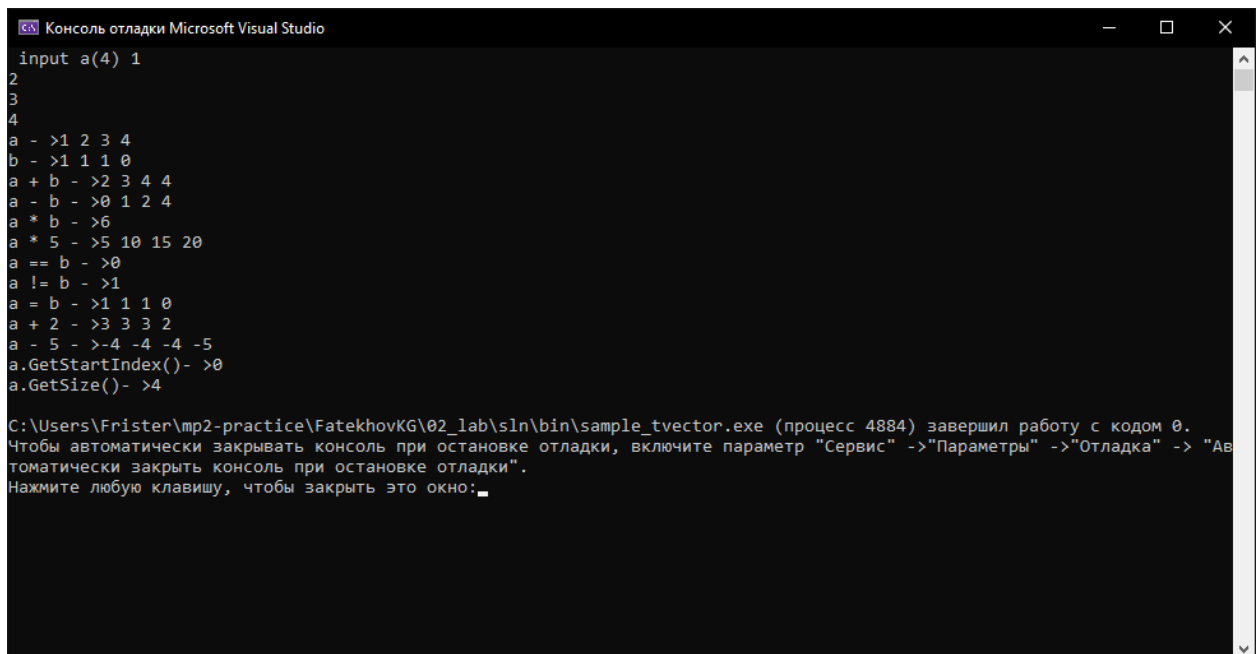


Рис. 1. Основное окно программы

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.



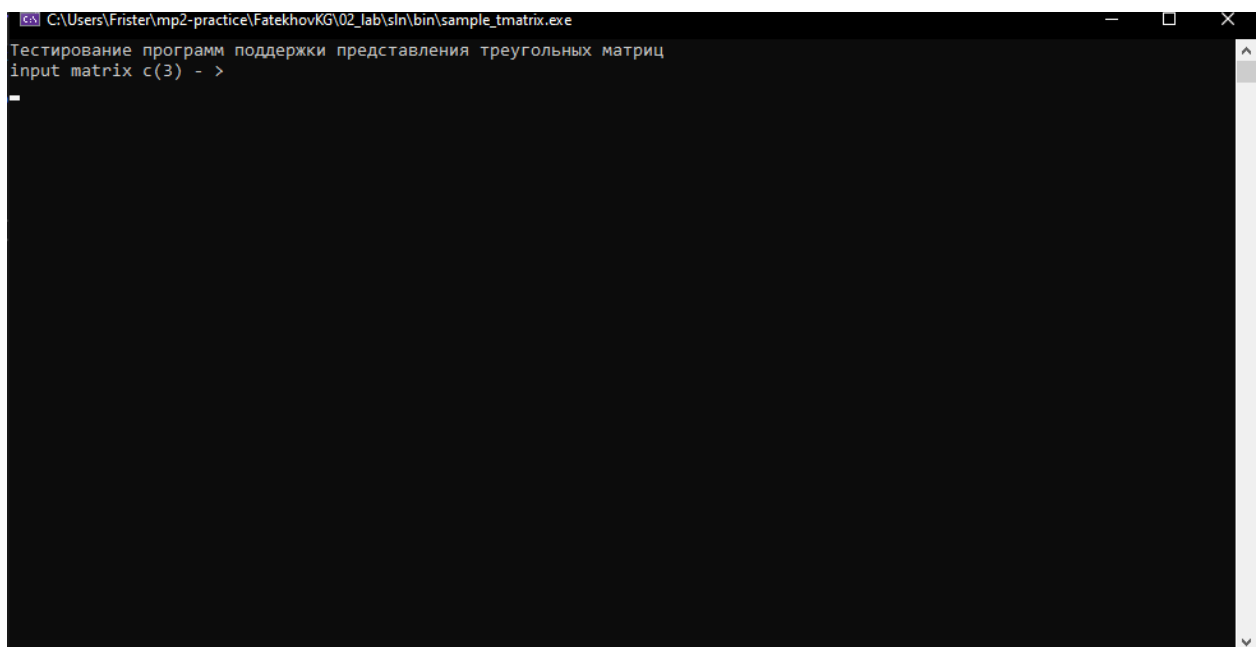
```
Консоль отладки Microsoft Visual Studio
input a(4) 1
2
3
4
a - >1 2 3 4
b - >1 1 1 0
a + b - >2 3 4 4
a - b - >0 1 2 4
a * b - >6
a * 5 - >5 10 15 20
a == b - >0
a != b - >1
a = b - >1 1 1 0
a + 2 - >3 3 3 2
a - 5 - >-4 -4 -4 -5
a.GetStartIndex()- >0
a.GetSize()- >4

C:\Users\Frister\mp2-practice\FatekhovKG\02_lab\sln\bin\sample_tvector.exe (процесс 4884) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рис. 2. Основное окно программы

## 2.2 Приложение для демонстрации работы матриц

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



```
C:\Users\Frister\mp2-practice\FatekhovKG\02_lab\sln\bin\sample_tmatrix.exe
Тестирование программ поддержки представления треугольных матриц
input matrix c(3) - >
_
```

Рис. 3. 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).

```
Консоль отладки Microsoft Visual Studio
тестирование программы поддержки представления треугольных матриц
input matrix c(3) - >
2
3
4
5
6
Matrix a =
0 1 2 3 4
11 12 13 14
22 23 24
33 34
44
Matrix b =
0 100 200 300 400
1100 1200 1300 1400
2200 2300 2400
3300 3400
4400
b + b =
0 101 202 303 404
1111 1212 1313 1414
2222 2323 2424
3333 3434
4444
a - b =
0 -99 -198 -297 -396
-1000 -1100 -1200 -1300 -1400
-2178 -2277 -2376
-3267 -3366
-4355
a * b =
0 1100 5600 15800 34000
12100 39600 84800 150000
16800 126500 216600
168900 261100
193600
a == b = 0
a != b = 1
a = a - 1
0 1 2 3 4
11 12 13 14
22 23 24
33 34
44
C:\Users\Frister\wp2-practice\Fatekhov\G02_lab\bin\bin\sample_matrix.exe (процесс 13888) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.
```

Рис. 4. Основное окно программы

## 3 Руководство программиста

### 3.1 Описание алгоритмов

#### 3.1.1 Векторы

1. Начало работы указано в пункте [2.1](#)
2. Описание методов и полей класса указано в пункте [3.2.1](#)

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:

1 2 3

[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:

$V1 = (2 \ 2 \ 2)$

$V2 = (1 \ 1 \ 1)$

$V3 = V1 - V2 = (1 \ 1 \ 1)$

## **3.1.2 Матрицы**

1. Начало работы указано в пункте [2.2](#)
2. Описание методов и полей класса указано в пункте [3.2.2](#)

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

A = B = 1;

## 3.2 Описание программной реализации

### 3.2.1 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** – длина 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.

### 3.2.2 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:

**MaxPower** – длина множества.

**BitField** – битовое поле.

Методы:

**TSet(int mp);**

Назначение: конструктор с параметром.

Входные параметры:

**mp** – длина множества.

Выходные данные:

Множество.

**TSet(const TSet &s);**

Назначение: конструктор копирования.

Входные параметры:

**s** – ссылка на константное множество.

Выходные параметры:

Множество.

**TSet(const TBitField &bf);**

Назначение: конструктор преобразования типа.

Входные параметры:

**bf** – ссылка на константное битовое поле

Выходные данные:

Множество.

**operator TBitField();**

Назначение: преобразование типа к битовому полю.

**int GetMaxPower(void) const;**

Назначение: получение мощности множества.

Выходные параметры:

Мощность множества.

**void InsElem(const int Elem);**

Назначение: включение элемента в множество.

Входные параметры:

**Elem** – новый элемент множества.

**void DelElem(const int Elem);**

Назначение: удаление элемента из множества.

Входные параметры:

**Elem** – элемент множества.

**int IsMember(const int Elem) const;**

Назначение: проверка наличия элемента в множестве.

Входные параметры:

**Elem** – элемент множества.

Выходные параметры:

Результат проверки

**int operator== (const TSet &s) const;**

Назначение: операция сравнения на равенство.

Входные элементы:

**s** – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

**int operator!= (const TSet &s) const;**

Назначение: операция сравнения на неравенство.

Входные элементы:

**s** – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

**const TSet& operator=(const TSet &s) ;**

Назначение: операция присваивания.

Входные параметры:

**s** – ссылка на константное множество.

Выходные параметры:

Ссылка на множество.

**TSet operator+ (const int Elem) ;**

Назначение: операция объединения с элементом.

Входные параметры:

**Elem** – новый элемент множества.

Выходные параметры:

Множество.

**TSet operator- (const int Elem);**

Назначение: операция разницы с элементом.

Входные параметры:

**Elem** – элемент множества.

Выходные параметры:

Множество.

**TSet operator+ (const TSet &s);**

Назначение: операция объединения множеств.

Входные параметры:

**s** – ссылка на константное множество.

Выходные параметры:

Множество.

**TSet operator\* (const TSet &s);**

Назначение: операция пересечения множеств.

Входные параметры:

**s** – ссылка на константное множество.

Выходные параметры:

Множество.

**TSet operator~ (void);**

Назначение: операция дополнения.

Выходные параметры:

Множество.

**friend istream& operator>>(istream& istr, TSet& bf);**

Назначение: перегрузка операции потокового ввода.

Входные параметры:

**istr** – ссылка на поток ввода.

**bf** – ссылка на константное множество.

Выходные параметры:

Ссылка на поток ввода.

**ostream &operator<<(ostream &ostr, const TSet &bf);**

Назначение: перегрузка операции потокового вывода.

Входные параметры:

**ostr** — ссылка на поток вывода.

**bf** — ссылка на константное множество.

Выходные параметры:

Ссылка на поток вывода.

## 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. Информация из нейронных сетей, в частности «CHAT GPT»



# Приложения

## Приложение А. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
    if (len < 0) throw "length can't be negative";
    BitLen = len;
    MemLen = BitLen / (sizeof(TELEM) * 8) + 1;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = 0;
}

TBitField::TBitField(const TBitField& bf) {
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = bf.pMem[i];
}

TBitField::~TBitField() {
    delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const {
    return (n / (sizeof(TELEM) * 8));
}

TELEM TBitField::GetMemMask(const int n) const {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    return 1 << (n & (sizeof(TELEM) * 8 - 1));
}

int TBitField::GetLength(void) const {
    return BitLen;
}

void TBitField::SetBit(const int n) {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    pMem[GetMemIndex(n)] = (GetMemMask(n) | pMem[GetMemIndex(n)]);
}

void TBitField::ClrBit(const int n) {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    pMem[GetMemIndex(n)] &= (~GetMemMask(n));
}

int TBitField::GetBit(const int n) const {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    if (pMem[GetMemIndex(n)] & GetMemMask(n)) return 1; else return 0;
}

ostream& operator <<(ostream& ostr, const TBitField& bf) {

    for (int i = 0; i < bf.BitLen; i++)
        if (bf.GetBit(i) == 0)
            ostr << 0;
        else ostr << 1;
    ostr << endl;
    return ostr;
}

TBitField& TBitField::operator=(const TBitField& bf) {
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    delete[] pMem;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++)
        pMem[i] = bf.pMem[i];
}
```

```

        return *this;
    }

    int TBitField::operator==(const TBitField& bf) const // сравнение
    {
        if (BitLen != bf.BitLen)
            return 0;
        int k = 0;
        for (int i = 0; i < MemLen; i++)
            if (pMem[i] != bf.pMem[i]) {
                return 0;
            }
        return 1;
    }

    int TBitField::operator!=(const TBitField& bf) const // сравнение
    {
        return !((*this) == bf);
    }

    TBitField TBitField::operator|(const TBitField& bf) // операция "или"
    {
        int k;
        int z;
        int y;
        if (BitLen > bf.BitLen)
        {
            y = 0; // This is big
            k = BitLen;
            z = bf.BitLen;
        }
        else {
            y = 1; //bf is big
            k = bf.BitLen;
            z = BitLen;
        }
        TBitField a(k);
        for (int i = 0; i <= GetMemIndex(z); i++) a.pMem[i] = bf.pMem[i] | pMem[i];
        for (int i = (GetMemIndex(z) + 1); i < a.MemLen; i++) if (y == 1) a.pMem[i] =
bf.pMem[i]; else a.pMem[i] = pMem[i];
        return a;
    }

    TBitField TBitField::operator&(const TBitField& bf) // операция "и"
    {
        int k;
        int z;
        if (BitLen > bf.BitLen) {
            k = BitLen;
            z = bf.BitLen;
        }

        else
        {
            k = bf.BitLen;
            z = BitLen;
        }

        TBitField a(k);
        for (int i = 0; i <= GetMemIndex(k); i++) a.pMem[i] = pMem[i] & bf.pMem[i];
        return a;
    }
}

```

```

TBitField TBitField::operator~(void) // отрицание
{
    TBitField a(*this);
    for (int i = 0; i < (a.MemLen - 1); i++) a.pMem[i] = ~(a.pMem[i]);
    for (int i = ((a.MemLen - 1) * (sizeof(TELEM) * 8)); i < (a.BitLen); i++) {
        if (a.GetBit(i) == 1) a.ClrBit(i);
        else a.SetBit(i);
    }
    return a;
}

// ввод/вывод

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    int tmp;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> tmp;
        if ((tmp != 0) && (tmp != 1)) {
            throw "The bit cannot take such a value";
        }

        if (tmp == 0) {
            bf.ClrBit(i);
        }
        else {
            bf.SetBit(i);
        }
    }
    return istr;
}

```

## **Приложение Б. Реализация класса TSet**

```

TSet::TSet(int mp) : BitField(mp)
{
    MaxPower = mp;
}

// конструктор копирования
TSet::TSet(const TSet& s) : BitField(s.BitField)
{
    MaxPower = s.GetMaxPower();
}

// конструктор преобразования типа
TSet::TSet(const TBitField& bf) : BitField(bf)
{
    MaxPower = bf.GetLength();
}

TSet::operator TBitField()
{
    return BitField;
}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const // элемент множества?
{
    if ((Elem < MaxPower) & (Elem >= 0))
        if (BitField.GetBit(Elem) == 1) return 1;
        else return 0; else throw "Negative Elem";
}

```

```

}

void TSet::InsElem(const int Elem) // включение элемента множества
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.SetBit(Elem); else throw "Negative n";
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.ClrBit(Elem); else throw "Negative n";
}

// теоретико-множественные операции

TSet& TSet::operator=(const TSet& s) // присваивание
{
    MaxPower = s.MaxPower;
    BitField = s.BitField;
    return *this;
}

int TSet::operator==(const TSet& s) const // сравнение
{
    return (BitField == s.BitField);
}

int TSet::operator!=(const TSet& s) const // сравнение
{
    return !(*this == s);
}

TSet TSet::operator+(const TSet& s) // объединение
{
    TSet a(BitField | s.BitField);
    return a;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.InsElem(Elem);
    }
    else throw "Negative Elem";
    return a;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.DelElem(Elem);
    }
    else throw "Negative Elem";
    return a;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    TSet a(BitField & s.BitField);
    return a;
}

```

```

}

TSet TSet::operator~(void) // дополнение
{
    TSet a(~BitField);
    return a;
}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    for (int i = 0; i < s.MaxPower; i++) {
        s.DelElem(i);
    }
    cout << "Input your set (To finish, enter -1)" << endl;
    int i;

    while (1) {
        istr >> i;
        if (i == -1) {
            return istr;
        }
        if ((i < 0) || (i > s.MaxPower)) {
            throw "OUTOFRANGE";
        }
        s.InsElem(i);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод
{
    cout << s.BitField;
    return ostr;
}

```