

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:

«Векторы и матрицы»

Выполнил(а): студент группы
3822Б1ФИ2

_____ / Хохлов А.Д./
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	5
3 Руководство программиста	7
3.1 Описание алгоритмов	7
3.1.1 Векторы	7
3.1.2 Матрицы.....	9
3.2 Описание программной реализации	11
3.2.1 Описание класса TVec	11
3.2.2 Описание класса TMatrix.....	15
Заключение	18
Литература	19
Приложения	20
Приложение А. Реализация класса TVec	20
Приложение Б. Реализация класса TMatrix	22

Введение

В современном мире информационных технологий большую роль играют операции с большими объемами данных. Одной из важных операций является эффективная работа с матрицами. Верхнетреугольные квадратные матрицы играют важную роль в многих областях математики и информатики.

Знание и понимание их структуры и принципов хранения помогают оптимизировать использование памяти и увеличивать эффективность вычислений.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

1 Постановка задачи

Цель: Целью данной лабораторной работы является создание структуры хранения верхнетреугольных квадратных матриц на языке программирования C++. В рамках работы необходимо разработать классы TVec и TMatrix, которые будут предоставлять функциональность для работы с векторами и матрицами. Основной задачей является реализация основных операций с верхнетреугольными квадратными матрицами, таких как сложение, вычитание, умножение, а также копирование, сравнение.

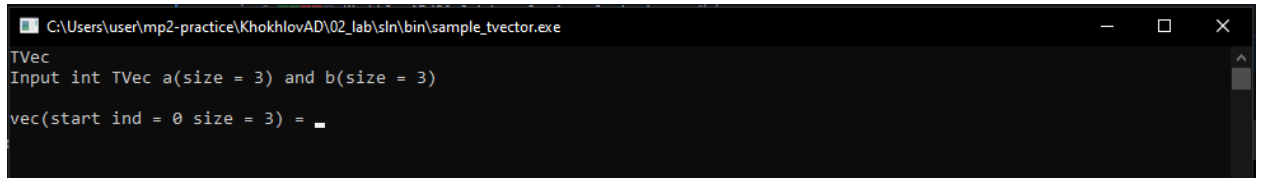
Задачи данной лабораторной работы:

- Разработка класса TVec, который будет предоставлять функциональность для работы с векторами.
- Реализация основных операций с векторами, основные арифметические операции и сравнение.
- Определение класса TMatrix, который будет предоставлять функциональность для работы с матрицами
- Реализация основных операций с матрицами, включая сложение, вычитание, умножение и сравнение.
- Проверка и демонстрация работы разработанных классов с помощью приложений.
- Написание отчета о выполненной лабораторной работе, включая описание алгоритмов, программной реализации и результатов работы.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

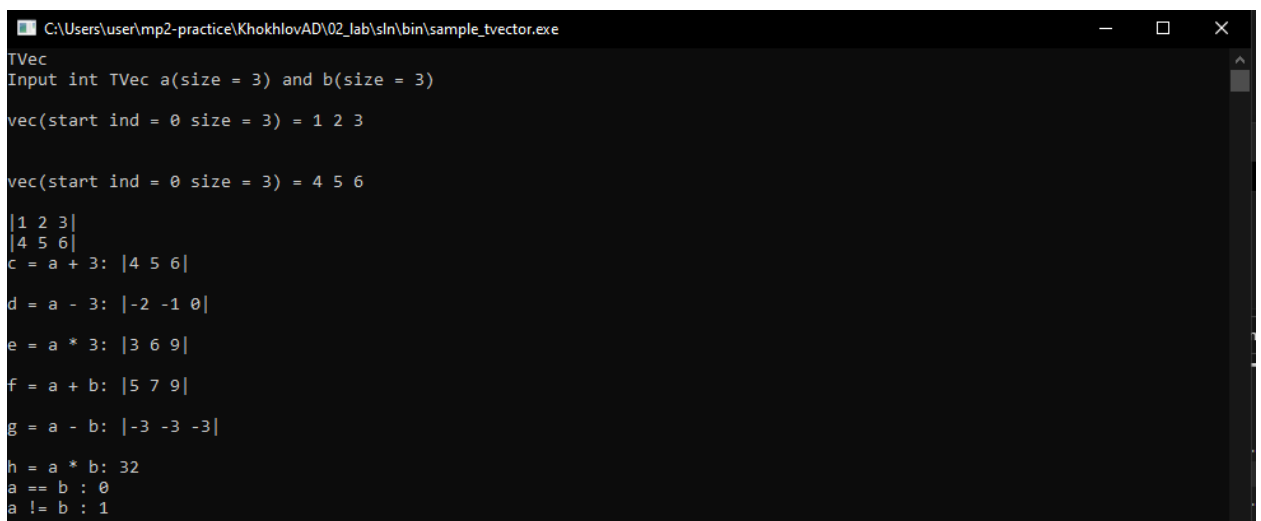
1. Запустите приложение с названием `sample_tvector.exe`. В результате появится окно, показанное ниже (рис. 1).



```
C:\Users\user\mp2-practice\KhokhlovAD\02_lab\sln\bin\sample_tvector.exe
TVec
Input int TVec a(size = 3) and b(size = 3)
vec(start ind = 0 size = 3) = _
```

Рис. 1. Основное окно программы

2. Это окно показывает работу основных функций работы с векторами (прибавление/вычитание константы, умножение на константу, сумма/разность векторов, произведение векторов, сравнение на равенство). Для продолжения введите значение векторов длины 3. В результате будет выведено (рис. 2). Для выхода нажмите любую клавишу.

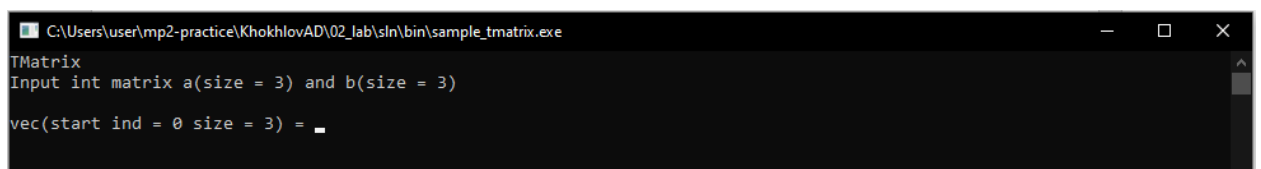


```
C:\Users\user\mp2-practice\KhokhlovAD\02_lab\sln\bin\sample_tvector.exe
TVec
Input int TVec a(size = 3) and b(size = 3)
vec(start ind = 0 size = 3) = 1 2 3
vec(start ind = 0 size = 3) = 4 5 6
| 1 2 3 |
| 4 5 6 |
c = a + 3: | 4 5 6 |
d = a - 3: | -2 -1 0 |
e = a * 3: | 3 6 9 |
f = a + b: | 5 7 9 |
g = a - b: | -3 -3 -3 |
h = a * b: 32
a == b : 0
a != b : 1
```

Рис. 2. Основное окно программы

2.2 Приложение для демонстрации работы матриц

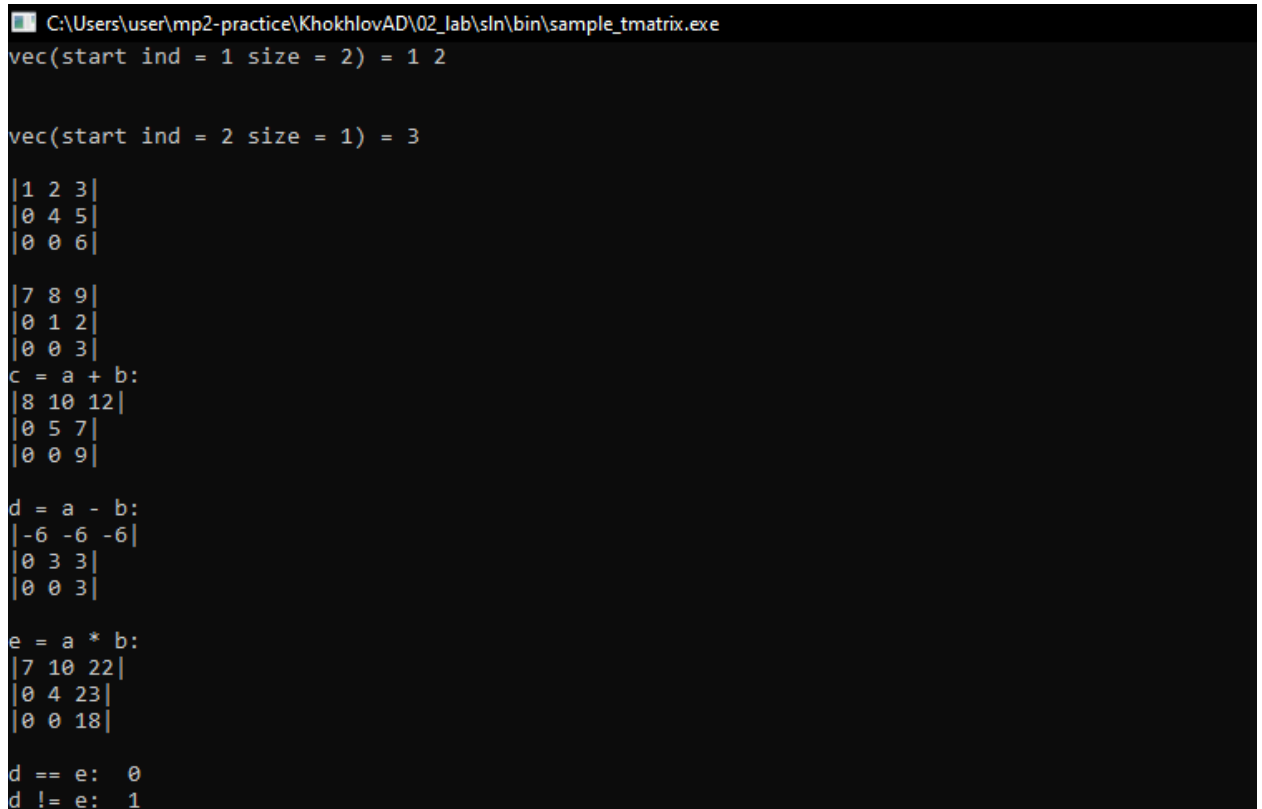
1. Запустите приложение с названием `sample_tmatrix.exe`. В результате появится окно, показанное ниже (рис. 3).



```
C:\Users\user\mp2-practice\KhokhlovAD\02_lab\sln\bin\sample_tmatrix.exe
TMatrix
Input int matrix a(size = 3) and b(size = 3)
vec(start ind = 0 size = 3) = _
```

Рис. 3. Основное окно программы

2. Введите значение верхнетреугольных квадратных матриц длины 3. Будет выведено следующее (рис. 4). Это окно показывает работу основных функций работы с матрицами (сложение, вычитание, произведение, сравнение на равенство). Для завершения программы нажмите любую клавишу.



```
C:\Users\user\mp2-practice\KhokhlovAD\02_lab\sln\bin\sample_tmatrix.exe
vec(start ind = 1 size = 2) = 1 2

vec(start ind = 2 size = 1) = 3

| 1 2 3 |
| 0 4 5 |
| 0 0 6 |

| 7 8 9 |
| 0 1 2 |
| 0 0 3 |
c = a + b:
| 8 10 12 |
| 0 5 7 |
| 0 0 9 |

d = a - b:
| -6 -6 -6 |
| 0 3 3 |
| 0 0 3 |

e = a * b:
| 7 10 22 |
| 0 4 23 |
| 0 0 18 |

d == e: 0
d != e: 1
```

Рис. 4. Основное окно программы

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Векторы

Вектор представляет собой структуру для хранения элементов одного типа данных и предоставляет эффективные операции для работы с матричными операциями. Внутри вектора элементы хранятся в виде массива, а также имеют стартовый индекс и количество элементов в векторе. Эта структура позволяет удобно выполнять операции над матрицами.

Если стартовый индекс не равен нулю, то все элементы от 0 до стартового индекса (не включительно) устанавливаются в нейтральный элемент (ноль).

Пример целочисленного вектора: стартовый индекс 1, размер 4: (0, 1, 2, 6, 12).

Вектор поддерживает операции сложения, вычитания и умножения с элементом того же типа данных, а также операции сложения, вычитания, скалярного произведения с вектором того же типа данных, операции индексации, сравнение на равенство (неравенство).

Операция сложения:

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

Пример:

$$V = \{1, 2, 3, 4, 5\}$$

Сложение с вектором:

$$V1 = \{1, 1, 1, 1, 1\}$$

$$V + V1 = \{2, 3, 4, 5, 6\}$$

Сложение с константой:

$$c = 2$$

$$V + c = \{3, 4, 5, 6, 7\}$$

Операция вычитания:

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

Пример:

$$V = \{1, 2, 3, 4, 5\}$$

Вычитание с вектором:

$$V1 = \{1, 1, 1, 1, 1\}$$

$$V - V1 = \{0, 1, 2, 3, 4\}$$

Вычитание с константой:

$$c = 2$$

$$V - c = \{-1, 0, 1, 2, 3\}$$

Операция умножения:

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

Пример:

$$V = \{1, 2, 3, 4, 5\}$$

Умножение на вектор:

$$V1 = \{1, 1, 1, 1, 1\}$$

$$V * V1 = 1*1 + 2*1 + 3*1 + 4*1 + 5*1 = 15$$

Умножение на константу:

$$c = 2$$

$$V * c = \{2, 4, 6, 8, 10\}$$

Операция индексации:

Операция индексации предназначена для получения элемента вектора. Если позиция меньше стартового индекса, возникает исключение.

Пример:

$$V = \{0, 1, 2, 3, 4\}$$

Получение элемента с индексом 1:

$$V[1] = 1$$

Операция сравнения на равенство:

Возвращает 1, если векторы равны поэлементно, включая стартовые индексы и размеры, и 0 в противном случае.

Пример:

$V = \{1, 2, 3, 4, 5\}$, $V1 = \{1, 2, 3, 4, 5\}$, $V2 = \{0, 1, 2, 3, 4\}$

$(V == V1) = 1$

$(V == V2) = 0$

Операция сравнения на неравенство:

Возвращает 0, если векторы равны поэлементно, включая стартовые индексы и размеры, и 1 в противном случае.

Пример:

$V = \{1, 2, 3, 4, 5\}$, $V1 = \{1, 2, 3, 4, 5\}$, $V2 = \{0, 1, 2, 3, 4\}$

$(V != V1) = 0$

$(V != V2) = 1$

3.1.2 Матрицы

Матрица представляет собой вектор векторов и является структурой для хранения элементов одного типа данных. Она представляется в виде массива векторов с указанием стартового индекса и количества элементов в матрице (число строк и столбцов, поскольку матрица квадратная и верхнетреугольная).

Пример целочисленной матрицы 3x3:

1	2	3
0	2	2
0	0	3

Данная матрица представляет собой массив трех векторов длины 3- i , со стартовым индексом i , где $i = \{0, 1, 2\}$.

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

Операция сложения:

Операция сложения определена для матриц того же типа. Реализовано сложением векторов вектора поэлементно с одинаковыми индексами.

Пример:

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 1 & 1 & 1 & 2 & 3 & 4 \\ 0 & 4 & 5 & + & 0 & 1 & 1 & = & 0 & 5 & 6 \\ 0 & 0 & 6 & 0 & 0 & 1 & 0 & 0 & 7 \end{array}$$

Операция вычитания:

Операция вычитания определена для матрицы того же типа. Реализовано вычитанием векторов вектора поэлементно с одинаковыми индексами.

Пример:

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 1 & 1 & 1 & 0 & 1 & 2 \\ 0 & 4 & 5 & - & 0 & 1 & 1 & = & 0 & 3 & 4 \\ 0 & 0 & 6 & 0 & 0 & 1 & 0 & 0 & 5 \end{array}$$

Операция умножения:

Операция умножения определена для матрицы того же типа. В результирующую матрицу записываются элементы по следующей формуле:

$$c_{ij} = \sum_{k=i}^j a_{ik} b_{kj}$$

(a,b – элементы матриц, i – номер строки, j – номер столбца)

Пример:

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 1 & 2 & 2 & 1 & 4 & 14 \\ 0 & 1 & 1 & * & 0 & 1 & 1 & = & 0 & 1 & 5 \\ 0 & 0 & 1 & 0 & 0 & 4 & 0 & 0 & 4 \end{array}$$

Операция индексации:

Операция индексации предназначена для получения элемента матрицы. Элемент матрицы представляет собой вектор-строку, и также можно извлекать элемент матрицы по индексу, так как для вектора также перегружена операция индексации.

Пример:

$$A = \begin{array}{ccc} 1 & 2 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array}$$

$A[0] = \{1, 2, 3\}$

$A[0][1] = 2$

Операция сравнения на равенство:

Возвращает 1, если матрицы равны поэлементно, включая стартовые индексы и размеры, и 0 в противном случае. Реализовано на перегрузке сравнения на равенство векторов. Сравнивает последовательно вектора.

Пример:

1	2	3		1	2	2
0	1	1	==	0	1	1 = <i>false</i>
0	0	1		0	0	4
1	2	3		1	2	3
0	1	1	==	0	1	1 = <i>true</i>
0	0	1		0	0	1

Операция сравнения на неравенство:

Возвращает 0, если матрицы равны поэлементно, включая стартовые индексы и размеры, и 1 в противном случае.

Пример:

1	2	3		1	2	2
0	1	1	!=	0	1	1 = <i>true</i>
0	0	1		0	0	4
1	2	3		1	2	3
0	1	1	!=	0	1	1 = <i>false</i>
0	0	1		0	0	1

3.2 Описание программной реализации

3.2.1 Описание класса TVec

```
class TVec {
protected:
    int size;
    int start_ind;
    T* pMem;
public:
    TVec(int size = 5, int start_ind = 0);
    TVec(const TVec<T>& vec);
    ~TVec();
    int GetSize() const noexcept;
    int GetStartIndex() const noexcept;
    T& operator[](const int index);
    T& operator[](const int index) const;
    bool operator==(const TVec<T>& vec) const;
    bool operator!=(const TVec<T>& vec) const;
    TVec operator*(const T& value);
    T operator*(const TVec<T>& vec);
    TVec operator+(const T& value);
```

```

TVec operator-(const T& value);
TVec operator+(const TVec<T>& vec);
TVec operator-(const TVec<T>& vec);
const TVec& operator=(const TVec<T>& vec);
friend ostream& operator<<(ostream& ostr, const TVec<T>& vec);
friend istream& operator>>(istream& istr, TVec<T>& vec);
};

```

Назначение: представление битового поля.

Поля:

size – длина вектора – максимальное количество элементов.

start_ind – начальный индекс вектора.

pMem – память для представления вектора.

Методы:

```
TVec(int size = 5, int start_ind = 0);
```

Назначение: конструктор с параметром.

Входные параметры:

size – длина вектора.

start_ind – стартовый индекс.

```
TVec(const TVec<T>& vec);
```

Назначение: конструктор копирования.

Входные параметры:

vec - вектор.

```
~TVec();
```

Назначение: деструктор.

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

Назначение: длина вектора.

Выходные параметры:

Длина вектора.

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

Назначение: стартовый индекс.

Выходные параметры:

Стартовый индекс.

T& operator[] (const int index) ;

Назначение: перегрузка оператора индексации.

Входные параметры:

index – номер элемента.

Выходные параметры:

Элемент вектора.

bool operator==(const TVec<T>& vec) const;

Назначение: перегрузка оператора сравнения на равенство.

Входные параметры:

vec – вектор.

Выходные параметры:

Результат сравнения.

bool operator!=(const TVec<T>& vec) const;

Назначение: перегрузка оператора сравнения на неравенство.

Входные параметры:

vec – вектор.

Выходные параметры:

Результат сравнения.

TVec operator*(const T& value) ;

Назначение: перегрузка операции произведения вектора на число.

Входные параметры:

value – константа.

Выходные параметры:

Результирующий вектор

T operator*(const TVec<T>& vec) ;

Назначение: перегрузка операции произведения векторов.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры:

Результат произведения.

TVec operator+(const T& value) ;

Назначение: перегрузка операции сложения вектора с числом.

Входные параметры:

value – константа

Выходные параметры:

Результирующий вектор.

TVec operator-(const T& value) ;

Назначение: перегрузка операции вычитания из вектора числа.

Входные параметры:

value – константа.

Выходные параметры:

Результирующий вектор.

TVec operator+(const TVec<T>& vec) ;

Назначение: перегрузка операции суммы векторов.

Входные параметры:

vec – ссылка на вектор

Выходные параметры:

Результирующий вектор.

TVec operator-(const TVec<T>& vec) ;

Назначение: перегрузка операции разности векторов.

Входные параметры:

vec – ссылка на вектор

Выходные параметры:

Результирующий вектор.

const TVec& operator=(const TVec<T>& vec) ;

Назначение: перегрузка оператора присваивания.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры:

Константная ссылка на результирующий вектор.

friend ostream& operator<<(ostream& ostr, const TVec<T>& vec) ;

Назначение: перегрузка потокового вывода.

Выходные параметры:

ostr – ссылка на поток вывода.

vec – ссылка на константный вектор.

Выходные параметры:

Ссылка на поток вывода.

```
friend istream& operator>>(istream& istr, TVec<T>& vec);
```

Назначение: перегрузка потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

vec – ссылка на константный вектор.

Выходные параметры:

Ссылка на поток ввода.

3.2.2 Описание класса TMatrix

```
class TMatrix : public TVec<TVec<T>>
{
public:
    TMatrix(int mn = 10);
    TMatrix(const TMatrix& mtrx);
    TMatrix(const TVec<TVec<T>>& mtrx);
    const TMatrix operator=(const TMatrix& mtrx);
    bool operator==(const TMatrix& mt) const;
    bool operator!=(const TMatrix& mt) const;
    TMatrix operator+(const TMatrix& mt);
    TMatrix operator-(const TMatrix& mt);
    TMatrix operator*(const TMatrix& mt);
    friend istream& operator>>(istream& istr, TMatrix<T>& mt);
    friend ostream& operator<<(ostream& ostr, const TMatrix<T>& mt);
};
```

Назначение: представление матриц.

Методы:

```
TMatrix(int mn = 10);
```

Назначение: конструктор с параметром.

Входные параметры:

mn – размер матрицы.

```
TMatrix(const TMatrix& mtrx);
```

Назначение: конструктор копирования.

Входные параметры:

mtrx – ссылка на константную матрицу.

```
TMatrix(const TVec<TVec<T>>& mtrx);
```

Назначение: конструктор преобразования типа.

Входные параметры:

mtrx – ссылка на константный вектор векторов.

```
const TMatrix operator=(const TMatrix& mtrx);
```

Назначение: перегрузка оператора присваивания.

Входные параметры:

mtrx – ссылка на константную матрицу.

Выходные параметры:

Константная результирующая матрица.

```
bool operator==(const TMatrix& mt) const;
```

Назначение: перегрузка оператора сравнения на равенство.

Входные параметры:

mt – ссылка на константную матрицу.

Выходные параметры:

Результат сравнения.

```
bool operator!=(const TMatrix& mt) const;
```

Назначение: перегрузка оператора сравнения на неравенство.

Входные параметры:

mt – ссылка на константную матрицу.

Выходные параметры:

Результат сравнения.

```
TMatrix operator+(const TMatrix& mt);
```

Назначение: перегрузка оператора суммы.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

```
TMatrix operator-(const TMatrix& mt);
```

Назначение: перегрузка оператора разности.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

```
TMatrix operator*(const TMatrix& mt);
```

Назначение: перегрузка оператора произведения.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

```
friend istream& operator>>(istream& istr, TMatrix<T>& mt);
```

Назначение: перегрузка операции потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

mt – ссылка на матрицу.

Выходные параметры:

Ссылка на поток ввода.

```
friend ostream& operator<<(ostream& ostr, const TMatrix<T>& mt);
```

Назначение: перегрузка операции потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

mt – ссылка на константную матрицу.

Выходные параметры:

Ссылка на поток вывода.

Заключение

В ходе выполнения работы "Векторы и матрицы" были изучены и практически применены концепции верхнетреугольных матриц.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы векторов и матриц

2. Была разработана программа, реализующая операции над векторами и матрицами.

В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами.

3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования векторов. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

Литература

1. Сысоев А.В., Алгоритмы и структуры данных, лекция 05, 5 октября.

Приложения

Приложение А. Реализация класса TVec

```
template <class T>
TVec<T>::TVec(int size, int start_ind) {
    if (size < 0)
        throw "invalid size";
    this->size = size;
    if (start_ind < 0)
        throw "invalid start index";
    this->start_ind = start_ind;
    pMem = new T[size];
}

template <class T>
TVec<T>::TVec(const TVec<T>& vec) {
    size = vec.size;
    start_ind = vec.start_ind;
    pMem = new T[size];
    for (int i = 0; i < size; i++)
        pMem[i] = vec.pMem[i];
}

template <class T>
TVec<T>::~TVec() {
    delete[] pMem;
}

template <class T>
int TVec<T>::GetSize()const noexcept {
    return size;
}

template <class T>
int TVec<T>::GetStartIndex()const noexcept {
    return start_ind;
}

template <class T>
T& TVec<T>::operator[](const int index) {
    return pMem[index];
}

template <class T>
const T& TVec<T>::operator[](const int index)const {
    return pMem[index];
}

template <class T>
bool TVec<T>::operator==(const TVec<T>& vec)const {
    if (start_ind != vec.start_ind)
        return false;
    if (size != vec.size)
        return false;
    for (int i = 0; i < size; i++)
        if (pMem[i] != vec.pMem[i])
            return false;
    return true;
}

template <class T>
```

```

bool TVec<T>::operator!=(const TVec<T>& vec) const {
    return !(*this == vec);
}

template <class T>
TVec<T> TVec<T>::operator*(const T& value) {
    TVec<T> tmp(*this);
    for (int i = 0; i < size; i++) {
        tmp.pMem[i] = pMem[i] * value;
    }
    return tmp;
}

template <class T>
T TVec<T>::operator*(const TVec<T>& vec) {
    if (start_ind != vec.start_ind)
        throw "invalid size";
    if (size != vec.size)
        throw "invalid start index";
    T res = T();
    for (int i = 0; i < size; i++)
        res += pMem[i] * vec.pMem[i];
    return res;
}

template <class T>
TVec<T> TVec<T>::operator+(const TVec<T>& vec) {
    if (start_ind != vec.start_ind)
        throw "invalid size";
    if (size != vec.size)
        throw "invalid start index";
    TVec<T> tmp(size, start_ind);
    for (int i = 0; i < size; i++)
        tmp.pMem[i] = pMem[i] + vec.pMem[i];
    return tmp;
}

template <class T>
TVec<T> TVec<T>::operator-(const TVec<T>& vec) {
    if (start_ind != vec.start_ind)
        throw "invalid size";
    if (size != vec.size)
        throw "invalid start index";
    TVec<T> tmp(size, start_ind);
    for (int i = 0; i < size; i++)
        tmp.pMem[i] = pMem[i] - vec.pMem[i];
    return tmp;
}

template <class T>
TVec<T> TVec<T>::operator+(const T& value) {
    TVec<T> tmp(size, start_ind);
    for (int i = 0; i < size; i++)
        tmp.pMem[i] = pMem[i] + value;
    return tmp;
}

template <class T>
TVec<T> TVec<T>::operator-(const T& value) {
    TVec<T> tmp(size, start_ind);
    for (int i = 0; i < size; i++)
        tmp.pMem[i] = pMem[i] - value;
    return tmp;
}

```

```

}

template <class T>
const TVec<T>& TVec<T>::operator=(const TVec<T>& vec) {
    if (*this == vec)
        return *this;
    if (size != vec.size)
    {
        size = vec.size;
        delete[] pMem;
        pMem = new T[size];
    }
    start_ind = vec.start_ind;
    for (int i = 0; i < size; i++)
        pMem[i] = vec.pMem[i];
    return *this;
}

friend ostream& operator<<(ostream& ostr, const TVec<T>& vec)
{
    ostr << "|";
    for (int i = 0; i < vec.start_ind; i++) {
        ostr << "0 ";
    }
    for (int i = 0; i < vec.size - 1; i++) {
        ostr << vec.pMem[i] << " ";
    }
    ostr << vec.pMem[vec.size-1];
    ostr << "|" << endl;
    return ostr;
}

friend istream& operator>>(istream& istr, TVec<T>& vec)
{
    cout << endl << "vec(start ind = " << vec.GetStartIndex() << "
size = " << vec.GetSize() << ") = ";
    for (int i = 0; i < vec.size; i++) {
        istr >> vec.pMem[i];
    }
    cout << endl;
    return istr;
}

```

Приложение Б. Реализация класса TMatrix

```

template <class T>
TMatrix<T>::TMatrix(int mn) : TVec<TVec<T>>>(mn) {
    if (mn < 0)
        throw "invalid size";
    for (int i = 0; i < mn; i++)
        pMem[i] = TVec<T>(mn - i, i);
}

template <class T>
TMatrix<T>::TMatrix(const TMatrix& mtrx) : TVec<TVec<T>>>((TVec<TVec<T>>>)mtrx)
{}

template <class T>
TMatrix<T>::TMatrix(const TVec<TVec<T>>>& mtrx) : TVec<TVec<T>>>(mtrx) {}

template <class T>
const TMatrix<T> TMatrix<T>::operator=(const TMatrix& mtrx) {
    return TVec<TVec<T>>>::operator=(mtrx);
}

```

```

}

template <class T>
bool TMatrix<T>::operator==(const TMatrix& mt)const {
    return TVec<TVec<T>> :: operator == (mt);
}

template <class T>
bool TMatrix<T>::operator!=(const TMatrix& mt)const {
    return TVec<TVec<T>> :: operator != (mt);
}

template <class T>
TMatrix<T> TMatrix<T>::operator+(const TMatrix& mt) {
    if (size != mt.size)
        throw "invalid size";
    return TVec<TVec<T>> :: operator + (mt);
}

template <class T>
TMatrix<T> TMatrix<T>::operator-(const TMatrix& mt) {
    if (size != mt.size)
        throw "invalid size";
    return TVec<TVec<T>> :: operator - (mt);
}

template <class T>
TMatrix<T> TMatrix<T>::operator*(const TMatrix& mt) {
    if (size != mt.size)
        throw "invalid size";
    TMatrix res(size);

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

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

    return res;
}

friend istream& operator>>(istream& istr, TMatrix<T>& mt)
{
    for (int i = 0; i < mt.size; i++) {
        istr >> mt.pMem[i];
    }
    return istr;
}

friend ostream& operator<<(ostream& ostr, const TMatrix<T>& mt)
{
    for (int i = 0; i < mt.size; i++) {
        ostr << mt.pMem[i];
    }
    return ostr;
}

```