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:

«Векторы и матрицы»

Выполнил(а): студент(ка) группы 3822Б1ФИ2
_____ / Холин К.И./
Подпись

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

Нижний Новгород
2023

Содержание

Введение

Ранее уже рассматривался шаблонный класс TVector. Теперь используем его для создания нового класса - TMatrix, предназначенный для работы с матрицами. Каждый элемент матрицы может быть найден на пересечении i-той строки и j-того столбца,где роль строки играет вектор,а столбец – индекс компоненты вектора. Использовать векторы очень удобно для создания матриц, поскольку это позволяет нам работать с матрицами как с двумерным массивом

1 Постановка задачи

Цель – реализовать шаблонные классы для работы с векторами TVector и матрицами TMatrix.

Задачи:

1. Разработать класс TVector, который должен поддерживать следующие операции: сложение, вычитание, копирование, равенство, неравенство.
2. Разработать класс TMatrix, который должен поддерживать следующие операции: сложение, вычитание, копирование матриц, равенство, неравенство.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

1. Запустите приложение с названием sample_TVector.exe. В результате появится окно, показанное на рисунке (рис. 1).

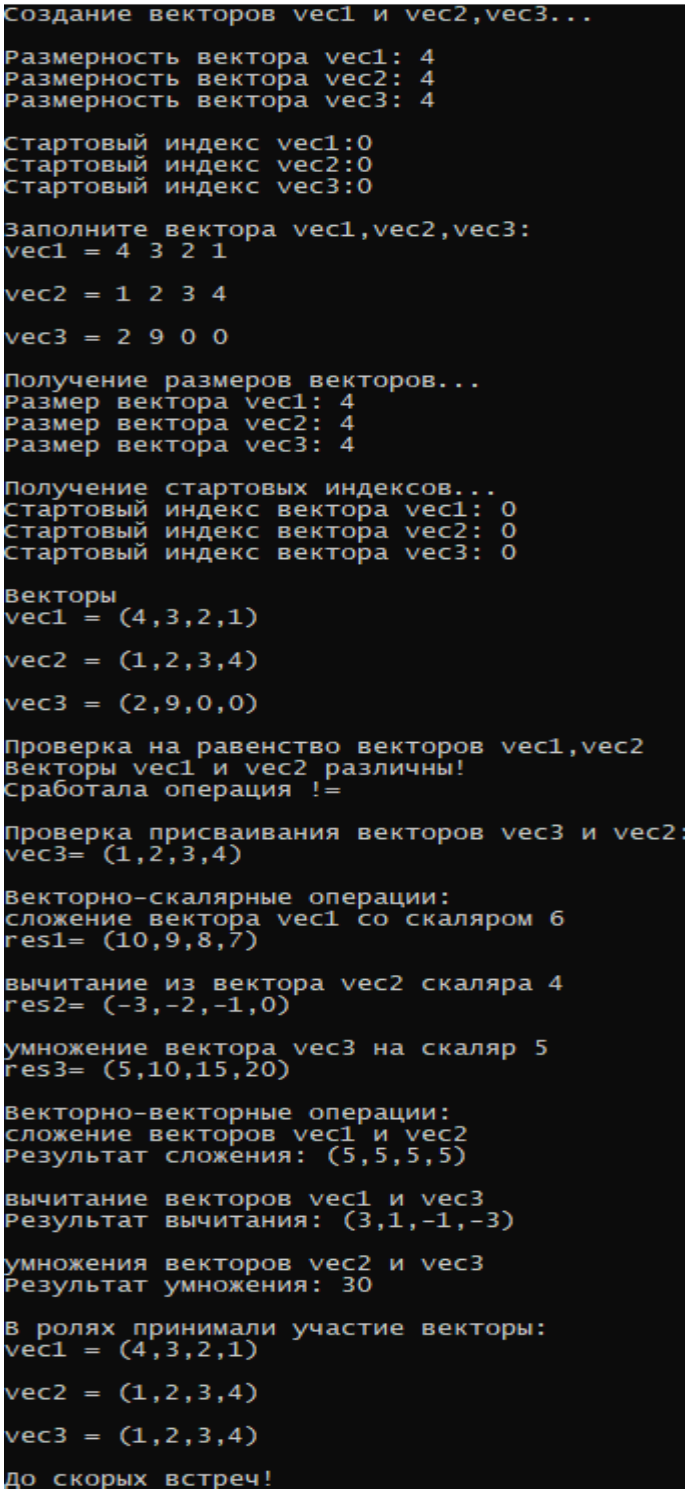

Рис. 1. Основное окно программы

2. В начале работы программы создаются 3 вектора (рис. 2).

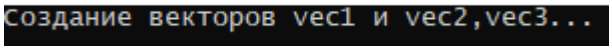

Рис. 2. Создание векторов

3. На следующем шаге выполняется заполнение векторов vec1, vec2, vec3 и их вывод данных (рис. 3).

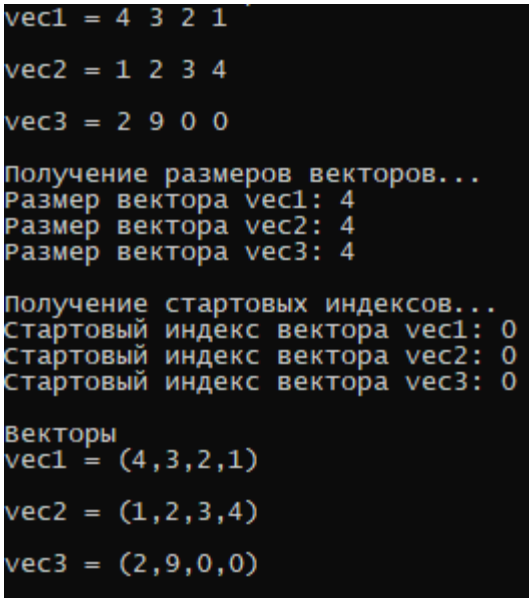

Рис. 3. Заполнение векторов и вывод

4. На рисунке ниже демонстрируется сравнение векторов vec1 и vec2 (рис. 4).

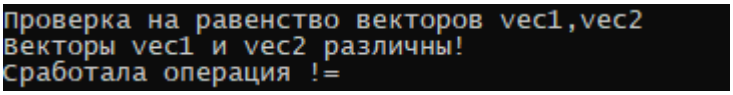

Рис. 4. Проверка на равенство векторов

5. Далее представлена работа операции присваивания векторов (рис. 5).

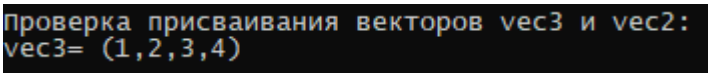

Рис. 5. присваивание векторов

6. Примеры работы векторно-скалярных операций (рис. 6).

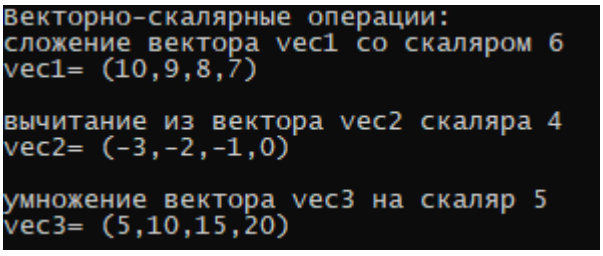

Рис. 6. Векторно-скалярные операции с векторами

7. Примеры векторно-векторных операций (рис. 7).

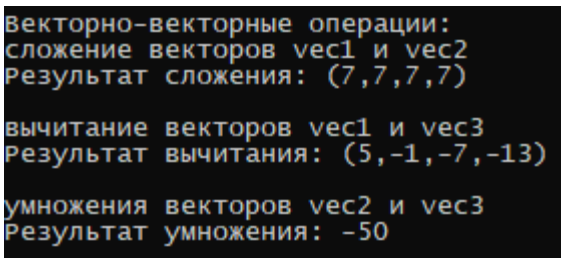

Рис. 7. Векторно-векторные операции с векторами

8. Для сравнения результатов векторы выводятся на экран (рис. 8).

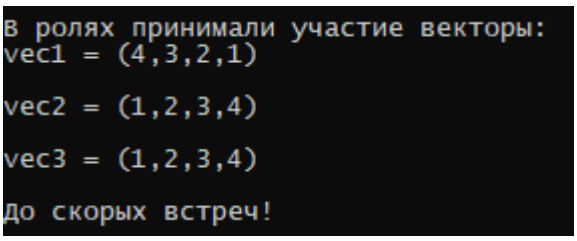

Рис. 8. Вывод

2.2 Приложение для демонстрации работы матриц

1. Запустите приложение с названием sample_TMatrix.exe. В результате появится окно, показанное ниже (рис. 9)

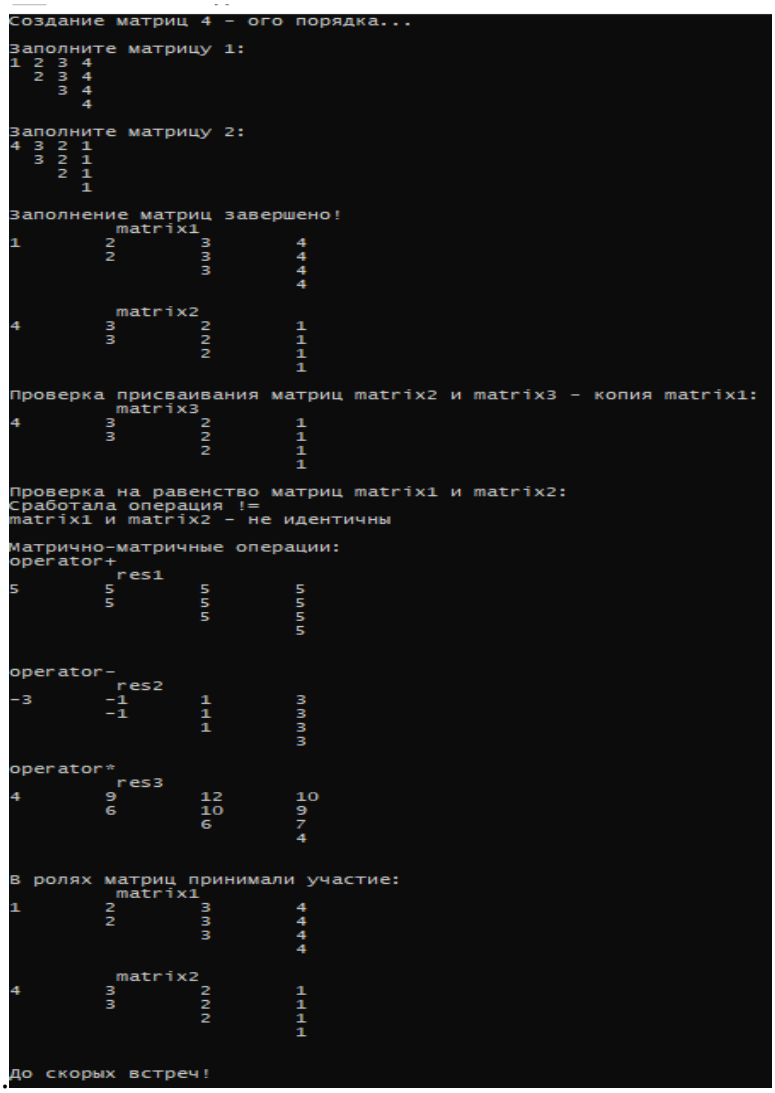

Рис. 9. Окно основной программы

2. В начале работы программы создаются матрицы указанного порядка (рис. 10).

Рис. 10. Создание матриц

3. На данном этапе пользователь вводит соответствующие значения элементов матриц, и результаты заполнения выводятся на экран (рис. 11).

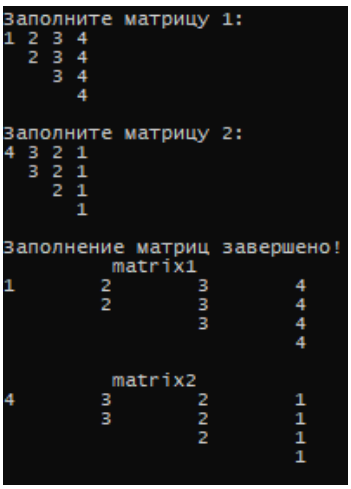

Рис. 11. Заполнение матриц и вывод

4. На рисунке ниже сравниваются две матрицы – matrix1 и matrix2 (рис. 12).

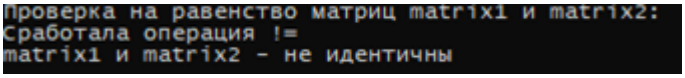

Рис. 12. Сравнение матриц

5. Примеры матрично-матричных операций (рис. 13).

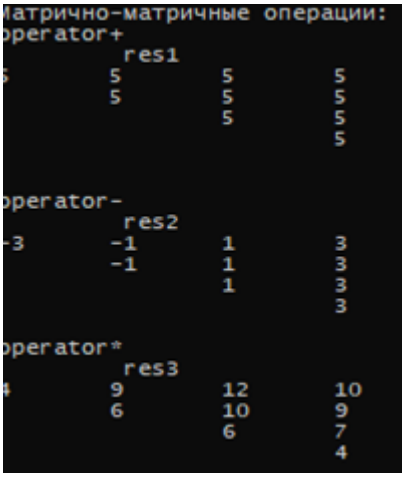

Рис. 13. Работа с матрицами

6. Для сравнения результатов матрицы выводятся на консоль (рис. 14).

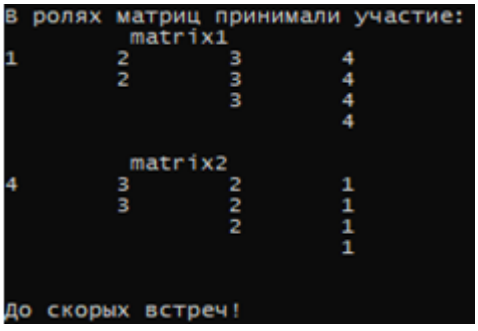

Рис. 14. Вывод

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Векторы

Векторы хранятся как динамический одномерный массив. Размер вектора задаётся количеством компонент. Компоненты вектора в памяти расположены, начиная с некоторого стартового индекса. Стартовый индекс – это индекс, начиная с которого можно получить доступ к компонентам вектора.

Рассмотрим базовые операции для работы с векторами.

1. Операция квадратные скобки. Нужна для получения доступа к компонентам вектора. В качестве примера возьмём вектор A размера 4 со стартовым индексом 1.

Index	0	1	2	3
Vector A	1	2	3	4

Известно, что выделенная память расположена от start_index до pos-start_index, где pos – это индекс, по которому мы хотим получить компоненту вектора.

```
Vector<int> A(5);
cout << A[0] << endl;
```

В результате будет получена компонента вектора с индексом 0.

2. Сложение векторов. Результатом сложения есть вектор C, который получается путём покомпонентного сложения векторов A и B.

Vector A	1	2	3	4
Vector B	3	0	9	-4

Результат:

Vector C	4	2	12	0

3. Вычитание векторов. Результатом сложения есть вектор C, который получается путём покомпонентного вычитания векторов A и B.

Vector A	1	2	3	4
Vector B	3	0	9	-4

Результат:

Vector C	-2	2	-6	8

4. Скалярное произведение. Результатом скалярного произведения есть действительное число, которое получается путём суммирования пар произведений соответствующих компонент векторов A и B.

Vector A	1	2	3	4
Vector B	3	0	9	-4

$(A, B) = \sum(A_i * B_i) = 1*3 + 2*0 + 3*9 + 4*(-4) = 14$

5. Умножение на скаляр. Результатом умножения на скаляр есть вектор C, который получается путём умножения каждой его компоненты на некоторую константу.

Scalar = 5

Vector C	4	2	12	0

Результат:

Scalar*Vector C	20	10	60	0

3.1.2 Матрицы

Матрица хранится как вектор векторов. Другими словами, массив массивов. Легко заметить, что матрицы полностью основаны на векторах. Каждый элемент такого вектора – это вектор. Обращение к элементам матрицам аналогично обращению к элементам двумерного массива, с которым мы уже умеем работать. Поэтому данная структура хранения эффективна и подходит для реализации матриц.

Рассматриваемый тип матриц – верхне-треугольные, где элементы ниже главной диагонали равны 0. Однако нулевые элементы хранить не будем.

Перейдём к рассмотрению базовых операций над матрицами.

1. Операция квадратные скобки. Для доступа к элементам матрицы необходимо дважды использовать данную операцию. В первый раз мы обращаемся к i - той строке(i - ый вектор вектора векторов),а во второй раз – к j-ому столбцу(j-тая компонента i - того вектора). Пример:

```
Matrix<int> A(4);
cout << A[1][2] << endl;
;
```

В результате буден получен элемент a_{12},где i = 1, j = 2.

2. Операция сложения. Операция сложения основана на операции сложении двух векторов. Для этого обращаемся к каждому вектору матрицы и, учитывая стартовый индекс вектора, получаем доступ к его компоненте и складываем с компонентой другого вектора другой матрицы.

Матрица 1

$$\begin{pmatrix} 3 & 2 & 3 & 4 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Вектор 1: (3, 2, 3, 4), стартовый индекс равен 0.

Вектор 2: (0, 2, 1, 4), стартовый индекс равен 1.

Вектор 3: (0, 0, 3, 3), стартовый индекс равен 2.

Вектор 4: (0, 0, 0, 4), стартовый индекс равен 3.

Матрица 2

$$\begin{pmatrix} 3 & 2 & 0 & 4 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Вектор 1: (3, 2, 0, 4), стартовый индекс равен 0.

Вектор 2: (0, 2, 1, 4), стартовый индекс равен 1.

Вектор 3: (0, 0, 3, 3), стартовый индекс равен 2.

Вектор 4: (0, 0, 0, 2), стартовый индекс равен 3.

Результирующая матрица

$$\begin{pmatrix} 6 & 4 & 3 & 8 \\ 0 & 4 & 2 & 8 \\ 0 & 0 & 6 & 6 \\ 0 & 0 & 0 & 6 \end{pmatrix}$$

Результирующий вектор 1: (6,4, 3, 8).

Результирующий вектор 2: (0, 4, 2, 8).

Результирующий вектор 3: (0, 0, 6, 6).

Результирующий вектор 4: (0, 0, 0, 6).

3. Операция вычитания. Из каждого элемента одной матрицы вычитается элемент другой матрицы соответственно. Для этого обращаемся к каждому вектору матрицы и, учитывая стартовый индекс вектора, получаем доступ к его компоненте и выполняем вычитание с компонентой другого вектора другой матрицы.

Матрица 1

$$\begin{pmatrix} 3 & 2 & 3 & 4 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Вектор 1: (3, 2, 3, 4), стартовый индекс равен 0.

Вектор 2: (0, 2, 1, 4), стартовый индекс равен 1.

Вектор 3: (0, 0, 3, 3), стартовый индекс равен 2.

Вектор 4: (0, 0, 0, 4), стартовый индекс равен 3.ё

Матрица 2

$$\begin{pmatrix} 4 & 2 & 0 & 2 \\ 0 & 6 & 1 & 8 \\ 0 & 0 & 7 & 3 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

Вектор 1: (4 2, 0, 2), стартовый индекс равен 0.

Вектор 2: (0, 6, 1, 8), стартовый индекс равен 1.

Вектор 3: (0, 0, 7, 3), стартовый индекс равен 2.

Вектор 4: (0, 0, 0, 3), стартовый индекс равен 3.

Результирующая матрица

$$\begin{pmatrix} -1 & 0 & 3 & 2 \\ 0 & 4 & 0 & -4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Результирующий вектор 1: (-1, 0, 3, 2).

Результирующий вектор 2: (0, 4, 0, -4).

Результирующий вектор 3: (0, 0, 0, 0).

Результирующий вектор 4: (0, 0, 0, 1).

4. Умножение матриц специального вида.

Общий принцип:

Первый элемент матрицы 1 соотносится с первым элементом первого столбца матрицы 2. Они перемножаются. Это и будет первый результирующий элемент результирующей матрицы в первой строке. Первые 2 элемента первой строки соотносятся с первыми двумя элементами второго столбца. Выполняется попарное умножение соответствующих элементов и результаты суммируются. Это и будет второй элемент первой строки результирующей матрицы. Обобщая по формуле:

$$\sum_{t=i}^{n-1} a_{it} * b_{tj}$$

$$i = 0, \ldots, n-1.\ j = 0, \ldots, k-1$$

Матрица 1

$$\begin{pmatrix} 3 & 2 & 3 & 4 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Матрица 2

$$\begin{pmatrix} 4 & 2 & 0 & 2 \\ 0 & 6 & 1 & 8 \\ 0 & 0 & 7 & 3 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

Результирующая матрица

$$\begin{pmatrix} 12 & 18 & 23 & 43 \\ 0 & 12 & 9 & 31 \\ 0 & 0 & 21 & 18 \\ 0 & 0 & 0 & 12 \end{pmatrix}$$

3.2 Описание программной реализации

3.2.1 Описание класса TVector

```
template <class Type> class TVector {
protected:
	int start_index;
	int size;
	Type* vector;
public:
	//#конструкторы и деструктор
	TVector(int size_ = 10, int start_index_ = 0);
	TVector(const TVector<Type>& obj);
	~TVector();

	//#свойства вектора
	int GetSize() const;
	int GetStart() const;
	Type& operator[](const int index);

	//#сравнение векторов
	int operator ==(const TVector<Type>& obj) const;
	int operator !=(const TVector<Type>& obj) const;

	TVector<Type>& operator=(const TVector<Type>& obj);
	//#векторно-скалярные операции
	TVector<Type> operator *(const Type& val);
	TVector<Type> operator +(const Type& val);
	TVector<Type> operator -(const Type& val);

	//#векторно-векторные операции
	TVector<Type> operator +(const TVector<Type>& obj);
	TVector<Type> operator -(const TVector<Type>& obj);
	Type operator*(const TVector<Type>& obj);



	//#ввод/вывод
	friend istream& operator>>(istream& istr, TVector<Type>& obj) {
		for (int i = 0; i < obj.GetSize(); i++) {
			istr >> obj.vector[i];
		}
		return istr;
	}

	friend ostream& operator<<(ostream& ostr,const TVector<Type>& obj) {
		cout << "(";
		for (int i = 0; i < obj.size; i++) {
			ostr << obj.vector[i];
			if (i == obj.size - 1) { continue; }
			cout << ",";
		}
		cout << ")" << endl;
		return ostr;
	}
};
```

Поля:

Size- размер вектора.

start_index – индекс, с которого выделяется память под вектор.

vector – память для хранения элементов вектора.

Методы:

```
TVector(int size_ = 10, int start_index = 0);
```

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

Входные параметры: **size_** – размер вектора, **start_index** – стартовый индекс.

Выходные параметры: отсутствуют.

```
TVector(const TVector<Type>& obj);
```

Назначение: копирование векторов.

Входные параметры: **obj -** ссылка на объект класса **TVector.**

Выходные параметры: отсутствуют.

```
~TVector();
```

Назначение: освобождение памяти.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

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

Назначение: получение размера вектора.

Входные параметры: отсутствуют.

Выходные параметры: **size** – размер вектора.

```
int GetStart()const;
```

Назначение: получение стартового индекса.

Входные параметры: отсутствуют.

Выходные параметры: **start_index** – стартовый индекс.

Type& operator[](const int index);

Назначение: получение элемента памяти vector.

Входные параметры: **index** – номер элемента.

Выходные параметры: элемент с номером **index - start_index**.

int operator ==(const TVector<Type>& obj)const;

Назначение: сравнение на равенство векторов.

Входные параметры: **obj** - ссылка на объект класса **TVector**.

Выходные параметры: целое число – 0 или 1.

int operator !=(const TVector<Type>& obj)const;

Назначение: сравнение на равенство векторов.

Входные параметры: **obj** - ссылка на объект класса **TVector**.

Выходные параметры: целое число – 0 или 1.

TVector<Type>& operator=(const TVector<Type>& obj);

Назначение: присваивание полей.

Входные параметры: **obj** - ссылка на объект класса **TVector**.

Выходные параметры: ссылка на объект себя.

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

Назначение: сложить вектор со скаляром.

Входные параметры: **val** - ссылка на скаляр.

Выходные параметры: новый объект класса **TVector** как результат сложения.

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

Назначение: вычесть из вектора скаляр.

Входные параметры: **val** – ссылка на скаляр.

Выходные параметры: новый объект класса **TVector** как результат вычитания.

TVector<Type> operator*(const Type& val);

Назначение: умножить вектор на скаляр.

Входные параметры: **val** - ссылка на скаляр.

Выходные параметры: новый объект класса **TVector** как результат умножения.

```
TVector<Type> operator+(const TVector<Type>& obj);
```

Назначение: сложить векторы.

Входные параметры: **obj** - ссылка на объект класса **TVector**.

Выходные параметры: новый объект класса **TVector** как результат сложения.

```
TVector<Type> operator-(const TVector<Type>& obj);
```

Назначение: вычесть один вектор из другого.

Входные параметры: **obj** - ссылка на объект класса **TVector**.

Выходные параметры: новый объект класса **TVector** как результат вычитания.

```
Type operator*(const TVector<Type>& obj);
```

Назначение: умножить один вектор на другой.

Входные параметры: **obj** – ссылка на объект класса **TVector**.

Выходные параметры: новый объект класса **TVector** как результат умножения.

```
friend istream& operator>>(istream& istr, TVector<Type>& obj);
```

Назначение: ввод элементов вектора.

Входные параметры: **istr** - ссылка на стандартный поток ввода,

obj – ссылка на объект класса **TVector**.

Выходные параметры: **istr** - ссылка на стандартный поток ввода.

```
friend ostream& operator<<(ostream& ostr,const TVector<Type>& obj)
```

Назначение: вывод элементов вектора.

Входные параметры: **ostr** - ссылка на стандартный поток вывода, **obj** – константная ссылка на объект класса **TVector**.

Выходные параметры: **ostr** - ссылка на стандартный поток вывода.

3.2.2 Описание класса TMatrix

```
template <class Type> class TMatrix : public TVector<TVector<Type>> {
public:
	//#конструкторы
	TMatrix(int mn = 10);
	TMatrix(const TMatrix<Type>& matr);
	TMatrix(const TVector<TVector<Type>>& v);

	const TMatrix<Type>& operator=(const TMatrix<Type>& matr);

	//#сравнение матриц
	int operator ==(const TMatrix<Type>& matr)const;
	int operator !=(const TMatrix<Type>& matr)const;

	//матричное-матричные операции
	TMatrix<Type> operator+(const TMatrix<Type>& matr);
	TMatrix<Type> operator-(const TMatrix<Type>& matr);
	TMatrix operator*(const TMatrix<Type>& matr);

	//#ввод/вывод
	friend istream& operator>>(istream& istr, TMatrix& obj) {
		for (int i = 0; i < obj.GetSize(); i++) {
			for (int k = obj.GetStart() + i; k < obj.GetSize(); k++) {
				cin >> obj[i][k];
			}
		}
		return istr;
	}
	friend ostream& operator<<(ostream& ostr, const TMatrix& obj) {
		for (int i = 0; i < obj.GetSize(); i++) {
			for (int j = 0; j < obj.GetStart() + i; j++) {
				ostr << "   ";
			}
			for (int k = obj.GetStart()+i; k < obj.GetSize(); k++) {
				ostr << obj.vector[i][k] << " ";
			}
			ostr << endl;

		}
		return ostr;
	}

};
```

Методы:

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

Назначение: выделение памяти под каждый вектор вектора векторов.

Входные параметры: **mn** – размерность матрицы.

Выходные параметры: отсутствуют.

```
TMatrix(const TMatrix<Type>& matr);
```

Назначение: копирование матриц.

Входные параметры: **matr** – ссылка на объект класса **TVector**.

Выходные параметры: отсутствуют.

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

Назначение: преобразует вектор векторов в матрицу.

Входные параметры: **`v`** - ссылка на объект класса **`TVector`**.

Выходные параметры: отсутствуют.

`const TMatrix<Type>& operator=(const TMatrix<Type>& matr);`

Назначение: присваивание матриц.

Входные параметры: **`matr`** - ссылка на объект класса TMatrix.

Выходные параметры: ссылка объект себя.

`int operator ==(const TMatrix<Type>& matr)const;`

Назначение: проверка матриц на равенство.

Входные параметры: **`matr`** - на объект класса TMatrix.

Выходные параметры: целое число – 0 или 1.

`int operator !=(const TMatrix<Type>& matr)const;`

Назначение: проверка матриц на неравенство.

Входные параметры: **`matr`** - ссылка на объект класса **`TMatrix`**.

Выходные параметры: целое число – 0 или 1.

`TMatrix<Type> operator+(const TMatrix<Type>& matr);`

Назначение: сложить матрицы.

Входные параметры: **`matr`** – ссылка на объект класса TMatrix.

Выходные параметры: новый объект класса TMatrix как результат сложения.

`TMatrix<Type> operator-(const TMatrix<Type>& matr);`

Назначение: вычесть из одной матрицы другую.

Входные параметры: **`matr`** - ссылка на объект класса **`TMatrix`**.

Выходные параметры: ссылка на объект класса **`TMatrix`** как результат вычитания.

`TMatrix operator*(const TMatrix<Type>& matr);`

Назначение: умножить матрицу на матрицу.

Входные параметры: **`matr`** – ссылка на объект класса **`TMatrix`**.

Выходные параметры: ссылка на объект класса **`TMatrix`** как результат умножения.

```
friend istream& operator>>(istream& istr, TMatrix& obj);
```

Назначение: ввод матриц.

Входные параметры: **istr** – ссылка на стандартный поток ввода,

obj – ссылка на объект класса **TMatrix**.

Выходные параметры: **istr** – ссылка на стандартный поток ввода.

```
friend ostream& operator<<(ostream& ostr, const TMatrix& obj);
```

Назначение: вывод матриц.

Входные параметры: **ostr** – ссылка на стандартный поток ввода,

obj – ссылка на объект класса **TMatrix**.

Выходные параметры: **ostr** – ссылка на стандартный поток вывода.

Заключение

Реализованы шаблонные классы TVector и TMatrix. В ходе лабораторной работы была выведена формула для умножения матриц специального вида, а также разработаны особые способы вывода и ввода верхне-треугольных матриц.

.

Литература

1. Гредасов Н.В, Линейная Алгебра – Издательство о Уральского университета. Редакционно-издательский отдел ИПЦ УрФУ 620049, 92 с.
2. Померанцев А., Векторы и матрицы – Российское Хемометрическое общество,33 разд.

Приложения

Приложение А. Реализация класса TVector

```
template <class Type>
TVector<Type>::TVector(int size_, int start_index_) {
	if (size_ <= 0 || size_ > INT_MAX) {
		throw Exeptions<int>(WRONG_SIZE, size_);
	}
	if (start_index_ < 0) {
		throw Exeptions<int>(WRONG_INDEX, start_index_);
	}
	size = size_;
	start_index = start_index_;
	vector = new Type[size];
	for (int i = 0; i < size; i++) {
		vector[i] = {};
	}
}

template <class Type>
TVector<Type>::TVector(const TVector<Type>& obj) {
	size = obj.size;
	start_index = obj.start_index;
	vector = new Type[size];
	for (int i = 0; i < obj.size; i++) {
		vector[i] = obj.vector[i];
	}
}

template<class Type>
TVector<Type>::~TVector() {
	delete[] vector;
	size = 0;
	start_index = 0;
}
template<class Type>
int TVector<Type>::GetSize() const {
	return size;
}

template<class Type>
int TVector<Type>::GetStart() const {
	return start_index;
}

template<class Type>
Type& TVector<Type>::operator[](const int index) {
	if (index < 0 || index >= size + start_index) {
		throw Exeptions<int>(WRONG_INDEX, index);
	}
	if (index < start_index) {
		throw Exeptions<int>(WRONG_INDEX, index);
	}
	return vector[index-start_index];
}
```

```
template<class Type>
TVector<Type>& TVector<Type>::operator=(const TVector<Type>& obj) {
	if (this == &obj) {
		return *this;
	}
	if (start_index != obj.start_index) {
		start_index = obj.start_index;
	}
	if (size != obj.size) {
		delete[] vector;
		size = obj.size;
		vector = new Type[size];
		for (int i = 0; i < size; i++) {
			vector[i] = {};
		}
	}
	for (int i = 0; i < size; i++) {
		vector[i] = obj.vector[i];
	}
	return *this;
}

template<class Type>
int TVector<Type>::operator==(const TVector<Type>& obj)const {
	if (size != obj.size) {
		return false;
	}
	if (start_index != obj.start_index) {
		return false;
	}
	for (int i = 0; i < size; i++) {
		if (vector[i] != obj.vector[i]) {
			return false;
		}
	}
	return true;
}

template<class Type>
int TVector<Type>::operator !=(const TVector<Type>& obj) const {
	return !(*this == obj);
}

template<class Type>
TVector<Type> TVector<Type>::operator*(const Type& val) {
	TVector<Type> tmp(*this);
	for (int i = 0; i < tmp.size; i++) {
		tmp[i] *= val;
	}
	return tmp;
}

template<class Type>
TVector<Type> TVector<Type>::operator+(const Type& val) {
	TVector<Type> tmp(*this);
	for (int i = 0; i < tmp.size; i++) {
		tmp[i] += val;
	}
	return tmp;
```

```
}

template<class Type>
TVector<Type> TVector<Type>::operator-(const Type& val) {
	TVector<Type> tmp(*this);
	for (int i = 0; i < tmp.size; i++) {
		tmp[i] -= val;
	}
	return tmp;
}

template<class Type>
TVector<Type> TVector<Type>::operator+(const TVector<Type>& obj) {
	if (start_index != obj.GetStart()){
 throw Exeptions<int>(WRONG_INDEX, start_index);
 }
	if (size != obj.size) {
		throw Exeptions<int>(WRONG_SIZE, size);
	}
	TVector<Type> result(*this);
	for (int i = 0; i < result.size; i++) {
		result.vector[i] = result.vector[i] + obj.vector[i];
	}
	return result;
}

template<class Type>
TVector<Type> TVector<Type>::operator-(const TVector<Type>& obj) {
	if (start_index != obj.GetStart())throw Exeptions<int>(WRONG_INDEX,
start_index);
	if (size != obj.size) {
		throw Exeptions<int>(WRONG_SIZE, size);
	}
	TVector<Type> result(*this);
	for (int i = 0; i < result.size; i++) {
		result.vector[i] = result.vector[i] - obj.vector[i];
	}
	return result;
}

template<class Type>
Type TVector<Type>::operator*(const TVector<Type>& obj) {
	if (start_index != obj.GetStart())throw Exeptions<int>(WRONG_INDEX,
start_index);
	if (size != obj.size) {
		throw Exeptions<int>(WRONG_SIZE, size);
	}
	Type result=0;
	for (int i = 0; i < size; i++) {
		result = result + (vector[i] * obj.vector[i]);
	}
	return result;
}
```

ПРИМЕР:

```
int main()
{

setlocale(LC_ALL, "rus");
cout << "Создание векторов vec1 и vec2,vec3..." << endl;
TVector<double>vec1(4),vec2(4),vec3(4),res1(1),
res2(1),res3(1),res4(1),res(1);
```

```
double scalar = 0.0;
cout << endl;

cout << "Размерность вектора vec1: " << vec1.GetSize() << endl;
cout << "Размерность вектора vec2: " << vec2.GetSize() << endl;
cout << "Размерность вектора vec3: " << vec3.GetSize() << endl;
cout << endl;

cout << "Стартовый индекс vec1:" << vec1.GetStart() << endl;
cout << "Стартовый индекс vec2:" << vec2.GetStart() << endl;
cout << "Стартовый индекс vec3:" << vec3.GetStart() << endl;
cout << endl;

cout << "Заполните вектора vec1,vec2,vec3: " << endl;
cout << "vec1 = ";
cin >> vec1;
cout << endl;

cout << "vec2 = ";
cin >> vec2;
cout << endl;

cout << "vec3 = ";
cin >> vec3;
cout << endl;

cout << "Получение размеров векторов..." << endl;
cout << "Размер вектора vec1: " << vec1.GetSize() << endl;
cout << "Размер вектора vec2: " << vec2.GetSize() << endl;
cout << "Размер вектора vec3: " << vec3.GetSize() << endl;
cout << endl;

cout << "Получение стартовых индексов..." << endl;
cout << "Стартовый индекс вектора vec1: " << vec1.GetStart() << endl;
cout << "Стартовый индекс вектора vec2: " << vec2.GetStart() << endl;
cout << "Стартовый индекс вектора vec3: " << vec3.GetStart() << endl;
cout << endl;

cout << "Векторы" << endl;
cout << "vec1 = " << vec1 << endl << "vec2 = "
<< vec2 << endl <<   "vec3 = " << vec3 << endl;

cout << "Проверка на равенство векторов vec1,vec2" << endl;
if (vec1 == vec2) {
    cout << "Векторы vec1 и vec2 одинаковые!" << endl;
    cout << "Сработала операция ==" << endl;
}
else if (vec1 != vec2) {
    cout << "Векторы vec1 и vec2 различны!" << endl;
    cout << "Сработала операция !=" << endl;
}
cout << endl;

cout << "Проверка присваивания векторов vec3 и vec2: " << endl;
vec3 = vec2;
cout << "vec3= " <<vec3 << endl;

cout << "Векторно-скалярные операции: " << endl;
cout << "сложение вектора vec1 со скаляром 6" << endl;
res1 = vec1 + 6;
cout << "res1= " << res1 << endl;

cout << "вычитание из вектора vec2 скаляра 4" << endl;
```

```
    res2 = vec2 - 4;
    cout << "res2= " << res2 << endl;

    cout << "умножение вектора vec3 на скаляр 5" << endl;
    res3 = vec3 * 5;
    cout << "res3= " << res3 << endl;

    cout << "Векторно-векторные операции: " << endl;
    cout << "сложение векторов vec1 и vec2" << endl;
    res1 = vec1 + vec2;
    cout << "Результат сложения: " << res1 << endl;

    cout << "вычитание векторов vec1 и vec3" << endl;
    res2 = vec1 - vec3;
    cout << "Результат вычитания: " << res2 << endl;

    cout << "умножения векторов vec2 и vec3" << endl;
    scalar = vec2 * vec3;
    cout << "Результат умножения: " << scalar << endl;
    cout << endl;

    cout << "В ролях принимали участие векторы: " << endl;
    cout << "vec1 = " << vec1 << endl << "vec2 = "
    << vec2 << endl <<     "vec3 = " << vec3 << endl;
    cout << "До скорых встреч!" << endl;


    return 0;
}
```

Приложение Б. Реализация класса TMatrix

```
template <class Type>
TMatrix<Type>::TMatrix(int mn): TVector<TVector<Type>>(mn) {
      for (int i = 0; i < mn; i++) {
            vector[i] = TVector<Type>(mn - i, i);
      }
}

template <class Type>
TMatrix<Type>::TMatrix(const              TMatrix<Type>&              matr):
TVector<TVector<Type>>(matr) {}

template<class Type>
TMatrix<Type>::TMatrix(const           TVector<TVector<Type>>&           v):
TVector<TVector<Type>>(v) {}



template<class Type>
int TMatrix<Type>::operator ==(const TMatrix<Type>& matr)const {
      return TVector<TVector<Type>>::operator==(matr);
}

template<class Type>
int TMatrix<Type>::operator !=(const TMatrix<Type>& matr)const {
      return TVector<TVector<Type>>::operator!=(matr);
}

template<class Type>
```

```
const TMatrix<Type>& TMatrix<Type>::operator=(const TMatrix<Type>& matr) {
	return TVector<TVector<Type>>::operator=(matr);
}


template<class Type>
TMatrix<Type> TMatrix<Type>::operator+(const TMatrix<Type>& matr) {
	return TVector<TVector<Type>>::operator+(matr);
}

template<class Type>
TMatrix<Type> TMatrix<Type>::operator-(const TMatrix<Type>& matr) {
	return TVector<TVector<Type>>::operator-(matr);
}

template<class Type>
TMatrix<Type> TMatrix<Type>::operator*(const TMatrix<Type>& matr) {
	if (size != matr.GetSize())throw Exeptions<int>(WRONG_SIZE, size);
	if  (start_index  !=  matr.GetStart())throw  Exeptions<int>(WRONG_INDEX,
start_index);

	TVector<TVector<Type>> result_matrix(GetSize());
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			for (int k = GetStart()+i; k < GetStart() + j + 1; k++) {
		result_matrix[i][j] += (*this).vector[i][k] * matr.vector[k][j];
			}
		}
	}
	return TMatrix(result_matrix);
}
```

ПРИМЕР:

```
int main()
{
	setlocale(LC_ALL, "rus");

	TMatrix<double>		matrix1(dim),		matrix2(dim),matrix3(matrix1),
res1(1),res2(1),res3(1);
	cout << "Создание матриц " << matrix1.GetSize() << " - ого порядка..."
<< endl;
	cout << endl;

	cout << "Заполните матрицу 1: " << endl;
	cin >> matrix1;
	cout << endl;

	cout << "Заполните матрицу 2: " << endl;
	cin >> matrix2;
	cout << endl;

	cout << "Заполнение матриц завершено!" << endl;
	for(int i =0; i < (matrix1.GetSize()/2)+1;i++){ cout << "    "; }
	cout << "matrix1";
	cout << endl;
	cout << matrix1 << endl;
	for (int i = 0; i < (matrix2.GetSize() / 2)+1; i++) { cout << "    "; }
	cout << "matrix2";
	cout << endl;
	cout << matrix2 << endl;
```

```
	cout << "Проверка присваивания матриц matrix2 и matrix3 - копия matrix1:" << endl;
	matrix3 = matrix2;
	for (int i = 0; i < (matrix3.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "matrix3";
	cout << endl;
	cout << matrix3 << endl;

	cout << "Проверка на равенство матриц matrix1 и matrix2:" << endl;
	if (matrix1 == matrix2) {
		cout << "Сработала операция ==" << endl;
		cout << "matrix1 и matrix2 - идентичны" << endl;
	}
	else if (matrix1 != matrix2){
		cout << "Сработала операция !=" << endl;
		cout << "matrix1 и matrix2 - не идентичны" << endl;
	}
	cout << endl;

	cout << "Матрично-матричные операции:" << endl;
	cout << "operator+" << endl;
	res1 = matrix1 + matrix2;
	for (int i = 0; i < (res1.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "res1";
	cout << endl;
	cout << res1 << endl;
	cout << endl;

	cout << "operator-" << endl;
	res2 = matrix1 - matrix2;
	for (int i = 0; i < (res2.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "res2";
	cout << endl;
	cout << res2 << endl;

	cout << "operator*" << endl;
	res3 = matrix1 * matrix2;
	for (int i = 0; i < (res3.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "res3";
	cout << endl;
	cout << res3 << endl;
	cout << endl;



	cout << "В ролях матриц принимали участие: " << endl;
	for (int i = 0; i < (matrix1.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "matrix1";
	cout << endl;
	cout << matrix1 << endl;
	for (int i = 0; i < (matrix2.GetSize() / 2) + 1; i++) { cout << "   "; }
	cout << "matrix2";
	cout << endl;
	cout << matrix2 << endl;
	cout << endl;
	cout << "До скорых встреч!" << endl;

    return 0;
}
```