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:

«Структуры хранения для матриц специального вида»

Выполнил(а): студент(ка) группы
_____ / Лысов И.М./
Подпись

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

Подпись

Нижний Новгород
2023

Оглавление

Введение

Треугольная матрица является частным случаем квадратной матрицы, где все элементы выше или ниже главной диагонали являются нулями. Верхняя треугольная матрица - это квадратная матрица, все элементы которой ниже главной диагонали являются нулями. Нижняя треугольная матрица представляет собой квадратную матрицу, все элементы которой выше главной диагонали равны нулям 1.

Треугольные матрицы обладают следующими свойствами:

1. Сумма треугольных матриц одного наименования есть треугольная матрица того же наименования; при этом диагональные элементы матриц складываются.
2. Произведение треугольных матриц одного наименования есть треугольная матрица того же наименования.
3. При возведении треугольной матрицы в целую положительную степень ее диагональные элементы возводятся в эту же самую степень.
4. При умножении треугольной матрицы на некоторое число ее диагональные элементы умножаются на это же самое число 2.

Векторные величины – величины, для характеристики которых указывается как числовое значение, так и направление в пространстве. Над векторами можно выполнять следующие операции:

1. Сложение и вычитание векторов.
2. Умножение векторов на скаляр.
3. Прибавление к координатам величину.
4. Скалярное произведение векторов.

Векторы и верхнетреугольные матрицы являются одними из главных элементов векторной алгебры и аналитической геометрии. Они применятся в физике, инженерии, биологии, медицине и других науках.

1 Цели и задачи

Цель: изучить векторы и верхнетреугольные матрицы. Получить навык практического применения шаблонов.

Задачи:

1. Изучить основные принципы работы с векторами и верхнетреугольными матрицами.
2. Разработать программу, которая будет реализовывать операции над векторами и верхнетреугольными матрицами.
3. Протестировать корректность выполнения программы на различных примерах и тестах.
4. Сделать выводы о проделанной работе.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

1. Запустить sample_vector.exe. В результате появится следующее окно: (рис. 1).

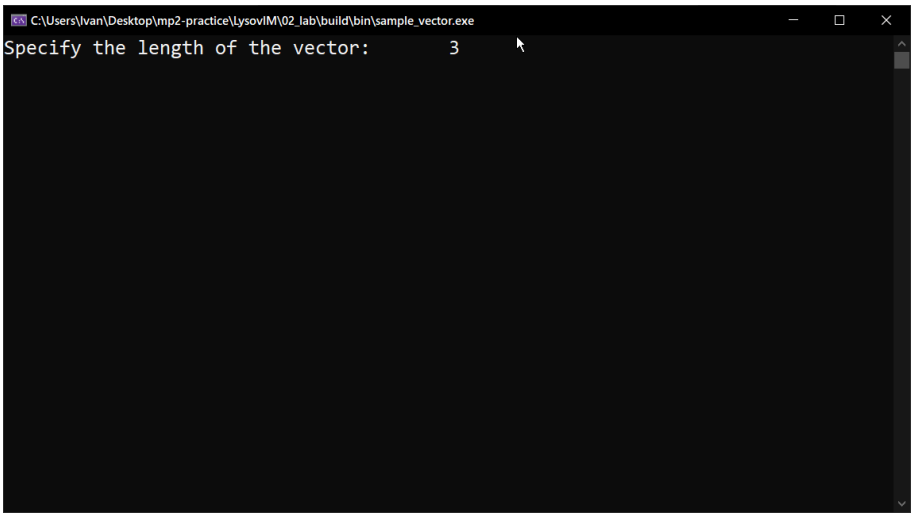

Рис. 1. Основное окно приложения векторов

На этом шаге необходимо ввести количество координат в векторе (len). Далее это число будет использоваться для второго вектора, как количество координат в векторе.

2. После ввода числа координат в векторе программа требует ввести соответствующие координаты для первого вектора (рис. 2).

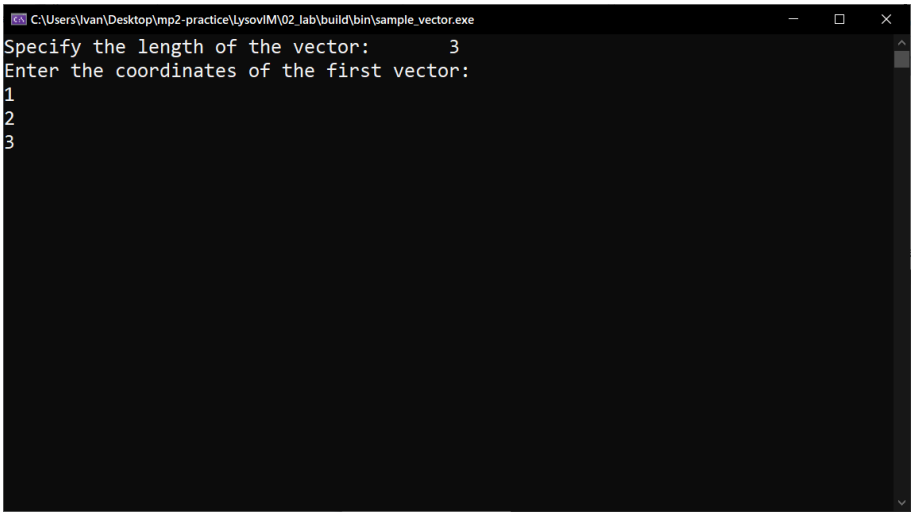

Рис. 2. Ввод координат первого вектора

3. После ввода координат первого вектора, программа требует ввести координаты второго вектора (рис. 3).

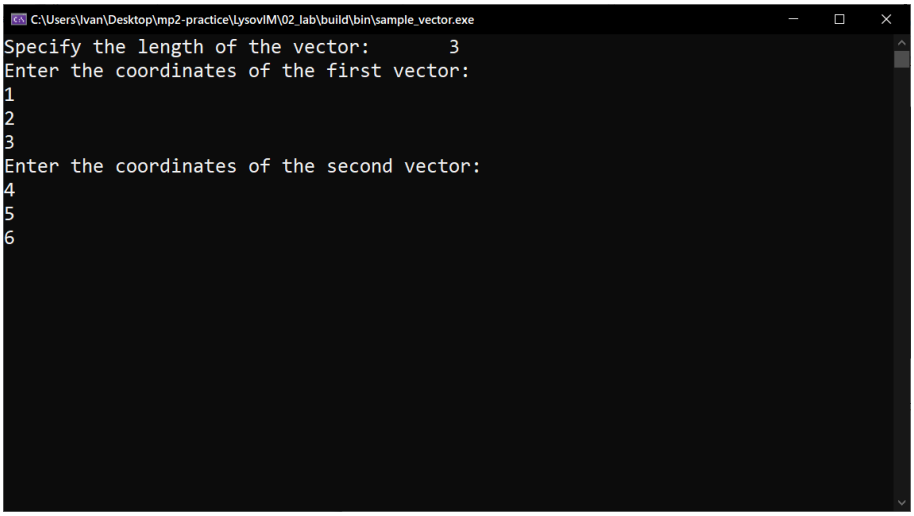

Рис. 3. Ввод координат второго вектора

4. После нажатия клавиши Enter программа выводит вектора и результаты применения операций над ними. Над векторами реализуются операции: сложение векторов, вычитание векторов, скалярное произведение, вычитание из вектора константы, умножение вектора на константу, проверка на равенство, проверка на неравенство (рис. 4).

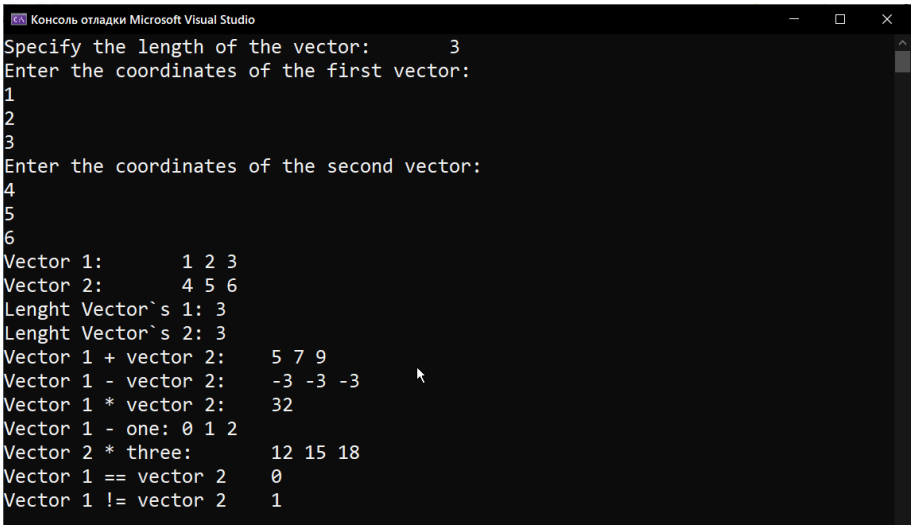

Рис. 4. Результат работы программы

2.2 Приложение для демонстрации работы матриц

1. Запустить sample_matrix.exe. В результате появится следующее окно: (рис. 5).

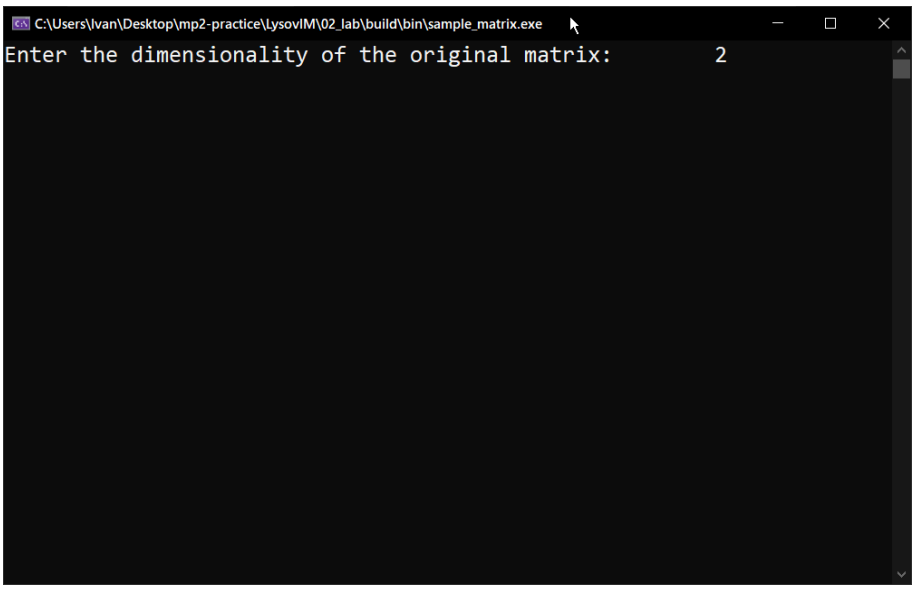

Рис. 5. Основное меню приложения для работы с матрицами

На этом шаге необходимо ввести размерность матрицы (dimension). Далее это число будет использоваться для второй матрицы, как её размерность.

2. После ввода размерности, программа требует ввести элементы первой верхнетругольной матрицы (рис. 6).

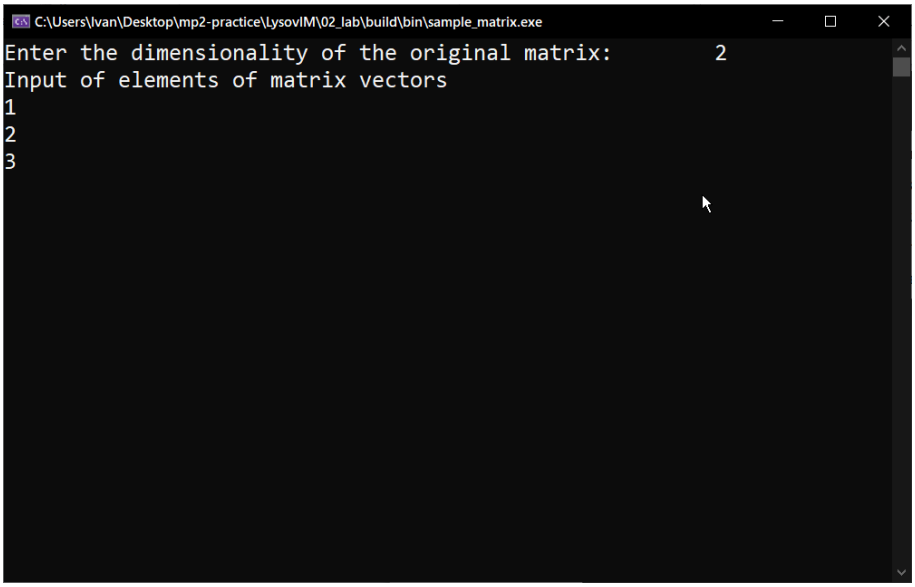

Рис. 6. Ввод элементов первой верхнетругольной матрицы

3. После ввода элементов первой матрицы, программа требует ввести элементы второй верхнетреугольной матрицы (рис. 7).

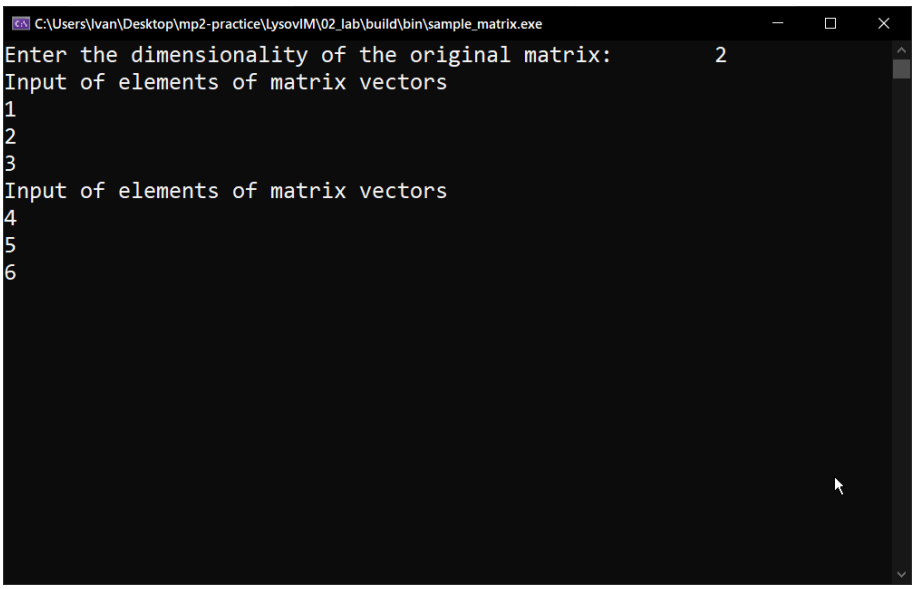

Рис. 7. Ввод элементов второй верхнетреугольной матрицы

4. После нажатия клавиши Enter программа выводит верхнетреугольные матрицы и результаты применения операций над ними. Над матрицами реализуются операции: сложение, умножение, проверка на равенство (рис. 8).

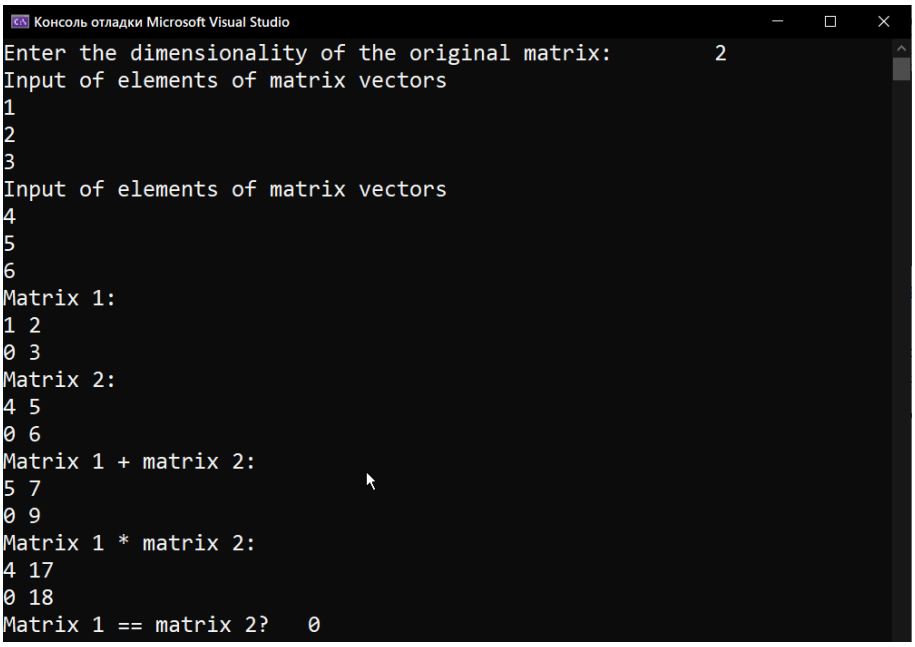

Рис. 8. Результат работы программы

3 Руководство программиста

3.1 Использованные алгоритмы

3.1.1. Вектор

Векторные величины – величины, для характеристики которых указывается числовое значение координат. Векторы удобны в представлении элементов и выполнения операций над ними.

Над векторами реализуются операции: сложение векторов, вычитание векторов, умножение векторов на скаляр, скалярное произведение векторов, проверка на равенство и неравенство.

- **Операция сложения (вычитания)**

Входные данные: 2 вектора.

Выходные данные: вектор, каждая координата, которого равна сумме (разности) элементов первого и второго вектора с одинаковыми индексами.

A	1	2	3
B	4	5	6
A+B	5	7	9
B-A	3	3	3

- **Операция умножения**

Входные данные: 2 вектора.

Выходные данные: вектор, каждый элемент которого равен элементов произведению элементов первого и второго вектора с одинаковыми индексами.

A	1	2	3
B	4	5	6
A*B	4	10	18

- **Операция добавление (вычитание) скаляра**

Входные данные: вектор.

Выходные данные: вектор, каждый элемент которого равен элементов сумме (разности) элементы первого и второго вектора с одинаковыми индексами.

A	1	2	3
A + 2	3	4	5
A - 3	-2	-1	0

- **Операция сравнения на равенство (неравенство)**

Входные данные: 2 вектора.

Выходные данные: операция сравнения выведет 1, если два вектора равны, или каждые их элементы совпадают, 0 в противном случае (в случае проверки на неравенства, наоборот).

A	1	1	1
B	1	1	1
A==B	1		
A!=B	0		

3.1.2. Матрица

Матрица удобный инструмент представления данных. В программе реализована верхнетреугольная матрица - это квадратная матрица, все элементы которой ниже главной диагонали являются нулями. Верхняя треугольная матрица поддерживает те же операции, что и обыкновенные матрицы (сложение, вычитание, умножение, проверка на равенство, проверка на неравенство).

В программе матрица реализуется с помощью векторов, записанных в строку.

- **Операция сложения**

Входные данные: 2 матрицы одного ранга.

Выходные данные: матрица, каждый элемент, которого элементов равен сумме элементов первой и второй матрицы с одинаковыми индексами.

$$A=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix},\ B=\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}\quad A+B=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix}+\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}=\begin{pmatrix}5&7&9\\0&7&9\\0&0&9\end{pmatrix}$$

- **Операция вычитания**

Входные данные: 2 матрицы.

Выходные данные: матрица, каждый элемент, которого элементов равен разности элементов первой и второй матрицы с одинаковыми индексами.

$$A=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix},\ B=\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}\quad A-B=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix}-\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}=\begin{pmatrix}-3&-3&-3\\0&-3&-3\\0&0&-3\end{pmatrix}$$

- **Операция умножения**

Входные данные: 2 матрицы.

Выходные данные: матрица, каждый элемент, которого элементов равен произведению элементов первой и второй матрицы с одинаковыми индексами.

$$A=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix},\ B=\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}\quad A*B=\begin{pmatrix}1&2&3\\0&2&3\\0&0&3\end{pmatrix}*\begin{pmatrix}4&5&6\\0&5&6\\0&0&6\end{pmatrix}=\begin{pmatrix}4&15&36\\0&10&30\\0&0&18\end{pmatrix}$$

- **Операция сравнения на равенство**

Входные данные: 2 матрицы.

Выходные данные: операция сравнения выведет 1, если две матрицы равны, или каждые их элементы совпадают, 0 в противном случае (в случае проверки на неравенства, наоборот).

$$A=\begin{pmatrix}1 & 2 & 3\\ 0 & 2 & 3\\ 0 & 0 & 3\end{pmatrix},\ B=\begin{pmatrix}4 & 5 & 6\\ 0 & 5 & 6\\ 0 & 0 & 6\end{pmatrix}$$

A==B	0
A!=B	1

3.2 Описание классов

3.2.1 Класс TVector

Объявление класса TVector:

```
class TVector
{
protected:
	int size;
	ValueType* pVector;
	int start_index;
public:
	TVector(int s = 5, int start_i = 0);
	TVector(const TVector& vector);
	~TVector();

	int getSize() const;
	int getStartIndex() const;

	ValueType& operator[](const int ind);
	int operator==(const TVector& vector) const;
	int operator!=(const TVector& vector) const;
	const TVector& operator=(const TVector& vector);
	TVector  operator+(const TVector& vector);
	TVector  operator-(const TVector& vector);
	double  operator*(const TVector& vector);

	TVector  operator+(const ValueType& value);
	TVector  operator-(const ValueType& value);
	TVector  operator*(const ValueType& value);
	friend istream& operator>>(istream& in, TVector& value)
	{
		for (int i = 0; i < value.size; ++i)
		{
			in >> value.pVector[i];
		}
		return in;
	}
	friend ostream& operator<<(ostream& out, const TVector& value)
	{
		for (int i = 0; i < value.size; ++i)
		{
			out << value.pVector[i] << " ";
		}
		out << endl;
		return out;
	}
```

```
};
```

Поля:

size – количество координат в векторе.

***pVector** – память для представления элементов вектора.

start_index – индекс первого необходимого элемента вектора.

Конструкторы:

- **TVector(int s = 5, int start_i = 0);**

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **s** – длина вектора, **start_i** – стартовый индекс.

Выходные параметры: отсутствуют.

- **TVector(const TVector& vector);**

Назначение: конструктор копирования.

Входные параметры: **vector** – экземпляр класса, на основе которого создаем новый объект.

Выходные параметры: отсутствуют.

Деструктор:

- **~TVector();**

Назначение: очистка выделенной памяти.

Входные и выходные параметры: отсутствуют.

Методы:

- **int getSize() const;**

Назначение: получение размера вектора.

Входные параметры: отсутствуют.

Выходные параметры: количество координат вектора.

- **int getStartIndex() const;**

Назначение: получение стартового индекса.

Входные параметры: отсутствуют.

Выходные параметры: стартовый индекс.

Операторы:

- **ValueType& operator[](const int ind);**

Назначение: перегрузка операции индексации.

Входные параметры: **ind** – индекс элемента.

Выходные параметры: элемент, который находится на **ind** позиции.

- **int operator==(const TVector& vector) const;**

Назначение: оператор проверки на равенство.

Входные параметры: **vector** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если не равны, 1 – если равны.

- **`int operator!=(const TVector& vector) const;`**

Назначение: оператор проверки на неравенство.

Входные параметры: **`vector`** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если равны, 1 – если не равны.

- **`const TVector& operator=(const TVector& vector);`**

Назначение: оператор присваивания.

Входные параметры **`vector`** – экземпляр класса, который присваиваем.

Выходные параметры: ссылка на (**`*this`**), уже присвоенный экземпляр класса.

- **`TVector  operator+(const TVector& vector);`**

Назначение: оператор сложения векторов.

Входные параметры: **`vector`** – вектор, который суммируем.

Выходные параметры: экземпляр класса, равный сумме двух векторов.

- **`TVector  operator-(const TVector& vector);`**

Назначение: оператор разности двух векторов.

Входные параметры: **`vector`**– вектор, который вычитаем.

Выходные параметры: экземпляр класса, равный разности двух векторов.

- **`double  operator*(const TVector& vector);`**

Назначение: оператор умножения векторов.

Входные параметры: **`vector`** – вектор, на который умножаем.

Выходные параметры: значение, равное скалярному произведению двух векторов.

- **`TVector  operator+(const ValueType& value);`**

Назначение: оператор сложения вектора и значения.

Входные параметры: **`value`** – элемент, с которым складываем вектор.

Выходные параметры: экземпляр класса, элементы которого на **`value`** больше.

- **`TVector  operator-(const ValueType& value);`**

Назначение: оператор вычитания вектора и значения.

Входные параметры: **`value`** – элемент, который вычитаем из вектора.

Выходные параметры: экземпляр класса, элементы которого на **`value`** меньше.

- **`TVector  operator*(const ValueType& value);`**

Назначение: оператор умножения вектора на значение.

Входные параметры: **`value`** – элемент, на который умножаем вектор.

Выходные параметры: экземпляр класса, элементы которого в **`value`** раз больше.

- **`friend istream& operator>>(istream& in, TVector& value)`**

Назначение: оператор ввода вектора.

Входные параметры: **`istr`** – поток ввода, **`value`** – ссылка на вектор, который вводим.

Выходные параметры: поток ввода.

- **friend ostream& operator<<(ostream& out, const TVector& value)**

Назначение: оператор вывода вектора.

Входные параметры: **ostr** – поток вывода, **value** – ссылка на вектор, который выводим.

Выходные параметры: поток вывода.

3.2.2 Класс TMatrix

Объявление класса TMatrix:

```
class TMatrix : public TVector<TVector<ValueType>>
{
public:
	TMatrix(int size);
	TMatrix(const TMatrix& matrix);
	TMatrix(const TVector<TVector<ValueType> >& matrix);

	int operator==(const TMatrix& matrix) const;
	int operator!=(const TMatrix& matrix) const;
	const TMatrix& operator= (const TMatrix& matrix);

	TMatrix  operator+(const TMatrix& matrix);
	TMatrix  operator-(const TMatrix& matrix);
	TMatrix operator*(const TMatrix& matrix);
	friend istream& operator>>(istream& in, TMatrix& matrix)
	{
		cout << "Input of elements of matrix vectors"<<endl;
		for (int i = 0; i < matrix.size; i++)
			in >> matrix.pVector[i];
		return in;
	}
	friend ostream& operator<<(std::ostream& ostr, const TMatrix<ValueType>&
m)
	{
		for (int i = 0; i < m.size; i++)
		{
			for (int j = 0; j < m.pVector[i].getStartIndex(); j++) {
				ostr << "0" << " ";
			}
			ostr << m.pVector[i];
		}
		return ostr;

	}
};
```

Конструкторы:

- **TMatrix(int size);**

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **size** – длина вектора.

Выходные параметры: отсутствуют.

- **TMatrix(const TMatrix& matrix);**

Назначение: конструктор копирования.

Входные параметры: **matrix** – экземпляр класса, на основе которого создаем новый объект.

Выходные параметры: отсутствуют.

- **TMatrix(const TVector<TVector<ValueType> >& matrix);**

Назначение: конструктор преобразования типов.

Входные параметры: **matrix** – ссылка на **TVector<TVector<T>>** - на объект, который преобразуем.

Выходные параметры: отсутствуют.

Операторы:

- **int operator==(const TMatrix& matrix) const;**

Назначение: оператор проверки на равенство.

Входные параметры: **matrix** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если не равны, 1 – если равны.

- **int operator!=(const TMatrix& matrix) const;**

Назначение: оператор проверки на неравенство.

Входные параметры: **matrix** – экземпляр класса, с которым сравниваем.

Выходные параметры: 0 – если равны, 1 – если не равны.

- **const TMatrix& operator= (const TMatrix& matrix);**

Назначение: оператор присваивания.

Входные параметры: **matrix** – экземпляр класса, который присваиваем.

Выходные параметры: ссылка на (***this**), уже присвоенный экземпляр класса.

- **TMatrix operator+(const TMatrix& matrix);**

Назначение: оператор сложения матриц.

Входные параметры: **matrix** – матрица, которую суммируем.

Выходные параметры: экземпляр класса, равный сумме двух матриц.

- **TMatrix operator-(const TMatrix& matrix);**

Назначение: оператор вычитания матриц.

Входные параметры: **matrix** – матрица, которую вычитаем.

Выходные параметры: экземпляр класса, равный разности двух матриц.

- **TMatrix operator*(const TMatrix& matrix);**

Назначение: оператор умножения матриц.

Входные параметры: **matrix** – матрица, которую умножаем.

Выходные параметры: экземпляр класса, равный произведению двух матриц.

- **friend istream& operator>>(istream& in, TMatrix& matrix);**

Назначение: оператор ввода матрицы.

Входные параметры: **istr** – поток ввода, **matrix** – ссылка на матрицу, которую вводим.

Выходные параметры: поток ввода.

friend ostream& operator<<(std::ostream& ostr, const TMatrix<ValueType>& m);

Назначение: оператор вывода матрицы.

Входные параметры: **ostr** – поток вывода, **m** – ссылка на матрицу, которую выводим.

Выходные параметры: поток вывода.

Заключение

В результате данной лабораторной работы были изучены теоретические основы векторов и матриц, а также принципы их использования в программировании. На основе полученных знаний была разработана программа, которая реализует операции над векторами (сложение, вычитание, добавление скаляра, умножение на скаляр, скалярное произведение векторов) и матрицами (сложение, вычитание, умножение). Проведенный анализ результатов показал, что использование векторов и матриц может быть очень полезным в решении определенных задач. Они позволяют эффективно работать с большим количеством данных, путем использования операций над ними.

Литература

1 Triangular Matrix [Triangular Matrix - Lower and Upper Triangular Matrix with Examples (turbopages.org)].

2 Треугольные, транспонированные и симметричные матрицы [https://portal.tpu.ru/SHARED/k/KONVAL/Sites/Russian_sites/1/10.htm].

Приложение

Реализация класса TVector:

```
template <typename ValueType>
int TVector<ValueType>::getSize() const
{
	return size;
}

template <typename ValueType>
int TVector<ValueType>::getStartIndex()  const
{
	return start_index;
}

template <typename ValueType>
TVector<ValueType>::TVector(int size, int startIndex) : size(size),
start_index(startIndex)
{
	pVector = new ValueType[size];
}

template <typename ValueType>
TVector<ValueType>::TVector(const TVector& vector) : size(vector.size),
start_index(vector.start_index)
{
	pVector = new ValueType[size];
	for (int i = 0; i < size; ++i)
	{
		pVector[i] = vector.pVector[i];
	}
}

template <typename ValueType>
TVector<ValueType>::~TVector()
{
	delete[] pVector;
}

template <typename ValueType>
ValueType& TVector<ValueType>::operator[](const int index)
{
	if (index < 0 || index >= size)
		throw ("Error: the index has gone out of range");
	return pVector[index];
}

template <typename ValueType>
int TVector<ValueType>::operator==(const TVector& vector) const
{
	if (this != &vector)
	{
		if ((size == vector.size) && (start_index == vector.start_index))
		{
			for (int i = 0; i < size; i++)
				if (pVector[i] != vector.pVector[i])
					return 0;
			return 1;
		}
		else
			return 0;
```

```
	}
	return 1;
}

template <typename ValueType>
int TVector<ValueType>::operator!=(const TVector& vector) const
{
	return !(*this == vector);
}

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

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator+(const TVector<ValueType>&
vector)
{
	if ((size != vector.size) || (start_index != vector.start_index))
		throw ("Error: vectors have different sizes");
	TVector result_vector(size, start_index);
	for (int i = 0; i < size; i++)
		result_vector.pVector[i] = pVector[i] + vector.pVector[i];
	return result_vector;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator-(const TVector<ValueType>&
vector)
{
	if ((size != vector.size) || (start_index != vector.start_index))
		throw ("Error: vectors have different sizes");
	TVector result_vector(size, start_index);
	for (int i = 0; i < size; i++)
		result_vector.pVector[i] = pVector[i] - vector.pVector[i];
	return result_vector;
}

template <typename ValueType>
double TVector<ValueType>::operator*(const TVector<ValueType>& vector)
{
	if ((size != vector.size) || (start_index != vector.start_index))
		throw ("Error: vectors have different sizes");
	double result_vector = 0;
	for (int i = 0; i < size; i++)
		result_vector = result_vector + (pVector[i] * vector.pVector[i]);
	return result_vector;
}
```

```
template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator+(const ValueType& value)
{
	TVector<ValueType> result_vector(*this);
	for (int i = 0; i < size; i++) {
		result_vector.pVector[i] = result_vector.pVector[i] + value;
	}
	return result_vector;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator-(const ValueType& value)
{
	TVector<ValueType> result_vector(*this);
	for (int i = 0; i < size; i++)
		result_vector.pVector[i] = result_vector.pVector[i] - value;
	return result_vector;
}

template <typename ValueType>
TVector<ValueType> TVector<ValueType>::operator*(const ValueType& value)
{
	TVector<ValueType> result_vector(*this);
	for (int i = 0; i < size; i++)
		result_vector.pVector[i] = result_vector.pVector[i] * value;
	return result_vector;
}
```

Реализация класса TMatrix:

```
template <typename ValueType>
TMatrix<ValueType>::TMatrix(int size) : TVector<TVector<ValueType>>(size)
{
	for (int i = 0; i < size; ++i)
	{
		pVector[i] = TVector<ValueType>(size - i, i);
	}
}

template <typename ValueType>
TMatrix<ValueType>::TMatrix(const TMatrix& matrix) :
TVector<TVector<ValueType>>(matrix) { }

template <typename ValueType>
TMatrix<ValueType>::TMatrix(const TVector<TVector<ValueType>>& matrix) :
TVector<TVector<ValueType> >(matrix) {}

template <typename ValueType>
int TMatrix<ValueType>::operator==(const TMatrix<ValueType>& matrix) const
{
	return TVector<TVector<ValueType> >::operator==(matrix);
}

template <typename ValueType>
int TMatrix<ValueType>::operator!=(const TMatrix<ValueType>& matrix) const
{
	return TVector<TVector<ValueType> >::operator!=(matrix);
}

template <typename ValueType>
const TMatrix<ValueType>& TMatrix<ValueType>::operator=(const
TMatrix<ValueType>& matrix)
{
	return TVector<TVector<ValueType> >::operator=(matrix);
```

```
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator+(const TMatrix<ValueType>&
matrix)
{
	return TVector<TVector<ValueType> >::operator+(matrix);
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator-(const TMatrix<ValueType>&
matrix)
{
	return TVector<TVector<ValueType> >::operator-(matrix);
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator*(const TMatrix& m)
{
	if (size != m.size)
		throw "Error: you cannot multiply sets with different dimensions";
	int size = this->getSize();
	TMatrix<ValueType> result(size);

	for (int k = 0; k < size; k++)
	{
		for (int j = k; j < size; j++)
		{
			ValueType summa = 0;
			for (int r = k; r <= j; r++)
			{
				summa = summa + (this->pVector[k][r - k] *
m.pVector[r][j - r]);
			}
			result.pVector[k][j - k] = summa;
		}
	}
	return result;
}
```