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 верхнетреугольных матриц»

Выполнил(а): студент(ка) группы 3822Б1ФИ1
_____ / Чистов А.Д./
Подпись

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

Подпись

Нижний Новгород
2023

Содержание

Введение....3
Постановка задачи....4
1 Руководство пользователя....5
1.1 Приложение для демонстрации работы векторов....5
1.2 Приложение для демонстрации работы матриц....6
2 Руководство программиста....7
2.1 Описание алгоритмов....7
2.1.1 Вектор....7
2.1.2 Матрица....8
2.2 Описание программной реализации....10
2.2.1 Описание класса TVector....10
2.2.2 Описание класса TMatrix....13
Заключение....16
Литература....17
Приложения....18
Приложение А. Реализация класса TVector....18
Приложение Б. Реализация класса TMatrix....19

Введение

Вектор – это математический объект, представляющие собой упорядоченный набор чисел, которые называются компонентами или координатами вектора. Векторы применяются в разных областях науки. Например, в физике для представления сил, скоростей, ускорений и других физических величин или в компьютерной графике для представления изображений, объектов и направлений движения и т.д. Мы же в данной лабораторной работе будем использовать векторы для представления верхнетреугольных матриц.

Верхнетреугольной матрица — квадратная матрица, в которой все элементы ниже главной диагонали равны нулю [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].

В целом можно сказать, что они являются важным инструментом в матричной алгебре и имеют множество приложений в науке, инженерии и других областях.

Постановка задачи

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

Задачи:

1. Исследовать тематическую литературу.
2. Реализовать класс TVector.
3. Реализовать класс TMatrix.
4. Провести тестирование разработанных классов для проверки их корректной работы.
5. Сделать выводы о проделанной работе.

1 Руководство пользователя

1.1 Приложение для демонстрации работы векторов

1. Запустите приложение с названием sample_vector.exe. В результате появится окно, показанное ниже, где вам нужно будет ввести размерность вектора (Рис. 1).

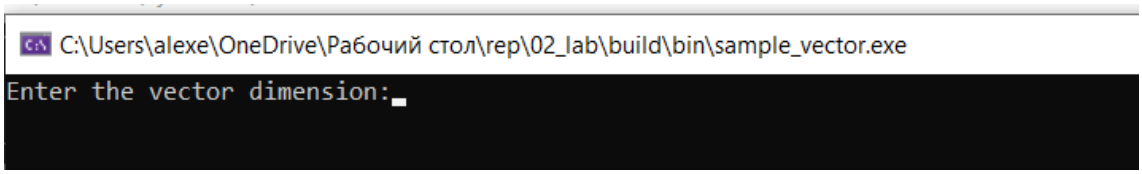

Рис. 1. Основное окно программы

2. Далее вам нужно будет ввести элементы двух векторов (Рис. 2).

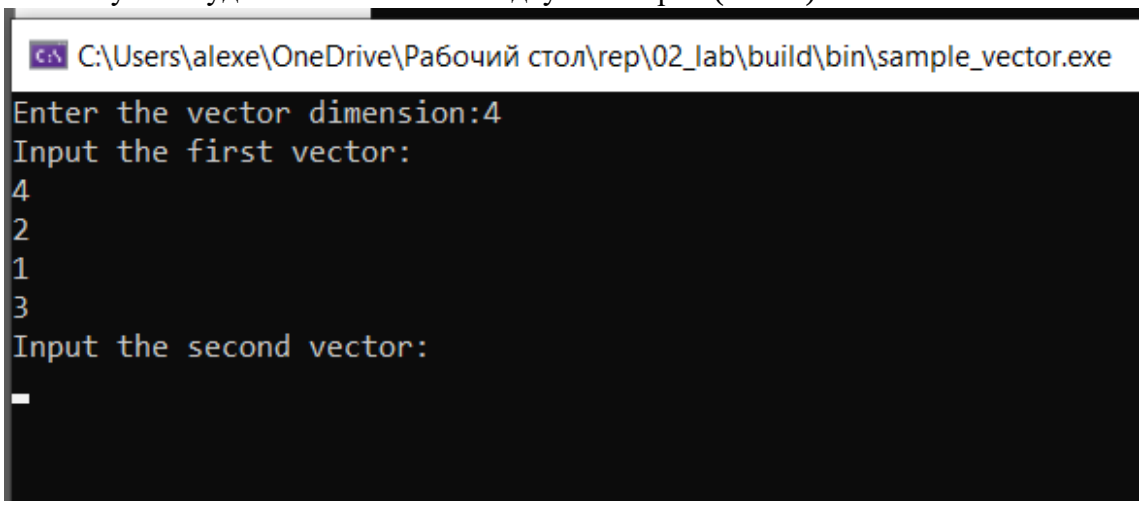

Рис. 2. Ввод векторов

3. Далее вы сможете наблюдать результат работы программы (Рис. 3).

Рис. 3. Результат работы программы

1.2 Приложение для демонстрации работы матриц

1. Запустите приложение с названием sample_matrix.exe,где вам нужно будет ввести размерность матрицы. В результате появится окно, показанное ниже (Рис. 4).

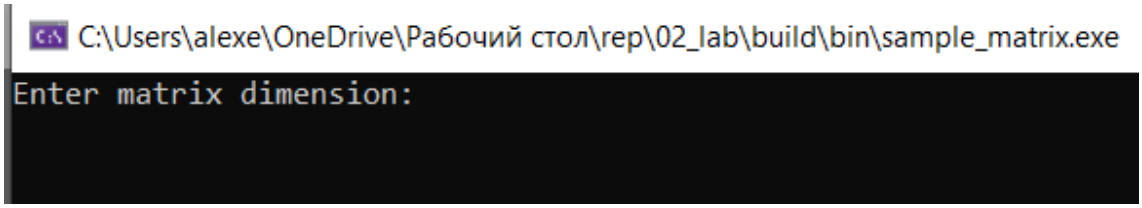

Рис. 4. Основное окно программы

2. Далее вам нужно будет ввести элементы двух матриц (Рис. 5).

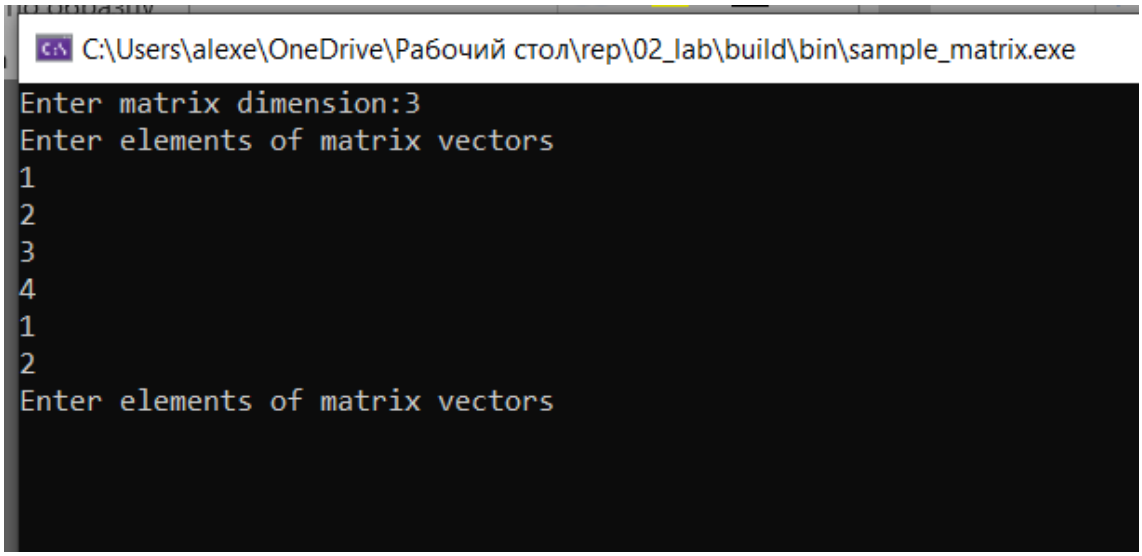

Рис. 5. Ввод матриц

3. Далее вы сможете наблюдать результат работы программы (Рис. 6).

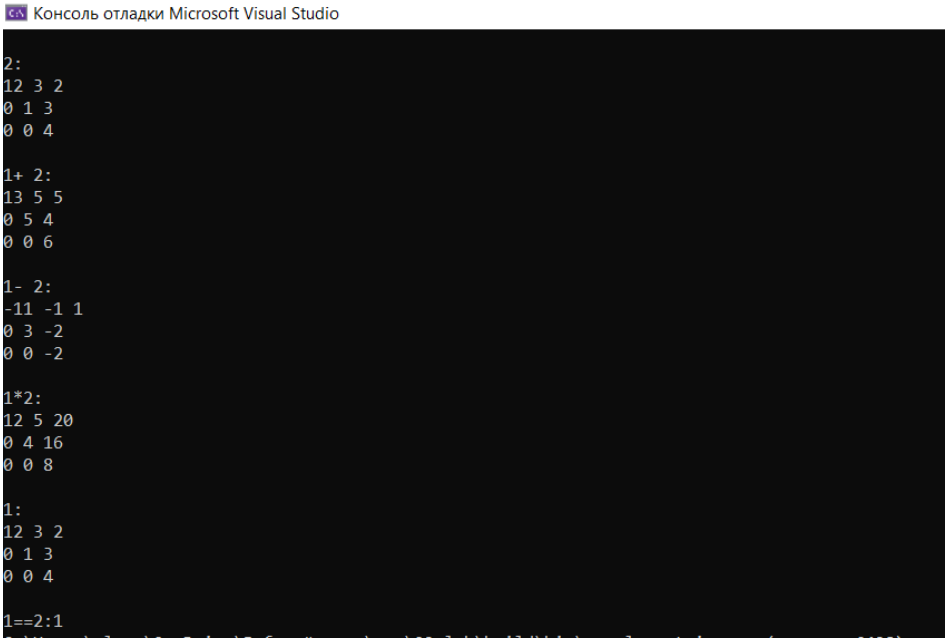

Рис. 6. Результат работы программы

2 Руководство программиста

2.1 Описание алгоритмов

2.1.1 Вектор

Вектор – структура хранения. Он хранит элементы одного типа данных. У каждого вектора есть свой стартовый индекс, с которого выделяется память для хранения элементов, и размер. Стартовый индекс необходим для удобного использования вектора внутри верхнетреугольной матрицы и представляет собой целое число, которое равно номеру строки матрицы, которую представляет собой этот вектор (это число принимает значение от 0 до значения размера матрицы не включительно).

Класс поддерживает следующие операции:

- **Операция сложения**

  Входные данные: вектора

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

  Пример:

  v1 = {1, 2, 3, 4}

  v2 = {5, 6, 7, 8}

  v1 + v2 = {6, 8, 10, 12}

- **Операция вычитания**

  Входные данные: вектора

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

  Пример:

  v1 = {1, 2, 3, 4,5}

  v2 = {2, 1, 2, 4, 5}

  v1 - v2 = {-1, 1, 1, 0,0}

- **Операция скалярного умножения**

  Входные данные: вектора

  Выходные данные: число, равное сумме попарных произведений соответствующих координат.

  Пример:

  v1 = {1, 2, 3, 4,5}

  v2 = {2, 1, 2, 4, 5}

  v1*v2 = 51

- **Операция добавление (вычитание) скаляра**

Входные данные: вектор и скаляр

Выходные данные: вектор, каждый элемент которого равен сумме(разности) элементов вектора и скаляра

Пример:

v1 = {1, 2, 3, 4,5}

v2 = {2, 1, 2, 4, 5}

v1+5 = {6, 7, 8, 9,10}

v2-10= {0, -1, 0, 2, 3}

- **Операция умножение на скаляр**

Входные данные: вектор и скаляр

Выходные данные: вектор, каждый элемент которого равен произведению элементов вектора и скаляра

Пример:

v1 = {1, 2, 3, 4,5}

v2 = {2, 1, 2, 4, 5}

v1+5 = {6, 7, 8, 9,10}

v2-10= {0, -1, 0, 2, 3}

2.1.2 Матрица

Верхнетреугольная матрица — это структура данных, которая представляет собой набор векторов, где каждый вектор содержит элементы одного типа, стартового индекса и количества элементов в матрице. Длина каждого вектора равна (n – i), где n – это размер матрицы (количество строк и столбцов), i – это номер соответствующего вектора-строки матрицы. Таким образом, в памяти не будут храниться нули ниже главной диагонали, что делает хранение такого типа матрицы эффективным с точки зрения затрат по памяти. Стартовый индекс соответствует номеру строки в матрице.

Класс поддерживает следующие операции:

- **Операция сложения**

Операция сложения верхнетреугольных матриц базируется на сложении соответствующих элементов двух матриц. Обращаемся к каждому вектору матрицы, учитывая их стартовый индекс, и складываем компоненты одного вектора с компонентами другого вектора и в результате формируем новую матрицу.

Пример:

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$A+B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 3 \\ 0 & 0 & 5 \end{pmatrix}$$

- **Операция вычитания**

Вычитание верхнетреугольных матриц с элементами одинакового типа осуществляется путем вычитания соответствующих элементов второй матрицы из элементов первой матрицы. Мы обращаемся к каждому вектору матрицы, учитывая их стартовый индекс. Аналогично операции сложения, это действие выполняется через работу с соответственными векторами, где вычитание одного вектора из другого формирует новую матрицу.

Пример:

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

$$A-B = \begin{pmatrix} -1 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & 0 & -1 \end{pmatrix}$$

- **Операция умножения**

Операция умножения определена для верхнетреугольных матриц с элементами одинакового типа. Первый элемент первой матрицы умножается на первый элемент первого столбца второй матрицы, и так далее. Выполняется попарное умножение соответствующих элементов и результаты суммируются. Результаты попарного умножения и суммирования элементов формируют результирующую матрицу.

Таким образом, общая формула имеет вид:

$$\sum_{k=i}^{j} a_{ik} * b_{kj}, (i = \overline{0, size-1}, j = \overline{i, size-1})$$

Пример:

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

$$A*B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

2.2 Описание программной реализации

2.2.1 Описание класса TVector

```
template <typename ValueType>
class TVector
{
protected:
      int size;
      ValueType* pVector;
      int startIndex;
public:
      TVector(int s = 5, int si = 0);
      TVector(const TVector& v);
      ~TVector();
      int GetSize() const;
      int GetStartIndex() const;
      ValueType& operator[](const int i);
      int operator==(const TVector& v) const;
      int operator!=(const TVector& v) const;
      const TVector& operator=(const TVector& v);

      TVector  operator+(const ValueType& val);
      TVector  operator-(const ValueType& val);
      TVector  operator*(const ValueType& val);

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

Назначение: представление вектора.

Поля:

Size – количество элементов вектора.

Start_Index – индекс первого необходимого элемента вектора.

pVec – память для представления элементов вектора.

Методы:

- **TVector (int s = 10, int index = 0);**

Назначение: конструктор по умолчанию и конструктор с параметрами.

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

- **`TVector (const TVector<T>& vec);`**
  Назначение: конструктор копирования.

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

- **`~TVector ();`**
  Назначение: освобождение выделенной памяти.

- **`int GetSize () const;`**

  Назначение: получение размера вектора.

  Выходные параметры: количество элементов вектора.

- **`int GetIndex() const;`**
  Назначение: получение стартового индекса.

  Выходные параметры: стартовый индекс.

  Операции:

- **`T& operator [] (const int index);`**
  Назначение: перегрузка операции индексации.

  Входные параметры: **`index`** – индекс (позиция) элемента.

  Выходные параметры: элемент, который находится на index позиции.

- **`int operator== (const TVector<T>& v) const;`**
  Назначение: оператор сравнения.

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

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

- **`int operator! = (const TVector<T>& v) const;`**
  Назначение: оператор сравнения.

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

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

- **`TVector operator*(const T& v);`**
  Назначение: оператор умножения вектора на значение.

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

  Выходные параметры: экземпляр класса, элементы которого в v раз больше.

- **`TVector operator+ (const T& v);`**
  Назначение: оператор сложения вектора и значения.

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

Выходные параметры: экземпляр класса, элементы которого на v больше.

- **TVector operator- (const T& v);**
  Назначение: оператор вычитания вектора и значения.

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

  Выходные параметры: экземпляр класса, элементы которого на v меньше.

- **TVector operator+ (const TVector<T>& v);**
  Назначение: оператор сложения векторов.

  Входные параметры: **v** – вектор, который суммируем.

  Выходные параметры: экземпляр класса, равный сумме двух векторов.

- **T operator*(const TVector<T>& v);**
  Назначение: оператор умножения векторов.

  Входные параметры: **v** – вектор, на который умножаем.

  Выходные параметры: значение, равное скалярному произведению двух векторов.

- **TVector operator- (const TVector<T>& v);**
  Назначение: оператор разности двух векторов.

  Входные параметры: **v** – вектор, который вычитаем.

  Выходные параметры: экземпляр класса, равный разности двух векторов.

- **const TVector& operator= (const TVector<T>& v);**
  Назначение: оператор присваивания.

  Входные параметры **v** – экземпляр класса, который присваиваем.

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

- **template<typename T> friend std:: ostream& operator>>(std::ostream& istr, const TVector<T>& v);**
  Назначение: оператор ввода вектора.

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

  Выходные параметры: поток ввода.

- **template<typename T> friend std::istream& operator<<(std::istream& ostr, TVector<T>& v);**
  Назначение: оператор вывода вектора.

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

Выходные параметры: поток вывода.

2.2.2 Описание класса TMatrix

```
template <typename ValueType>
class TMatrix : public TVector<TVector<ValueType>>
{
public:
	TMatrix(int size);
	TMatrix(const TMatrix& mt);
	TMatrix(const TVector<TVector<ValueType> >& mt);
	int operator==(const TMatrix& mt) const;
	int operator!=(const TMatrix& mt) const;
	const TMatrix& operator= (const TMatrix& mt);
	TMatrix  operator+(const TMatrix& mt);
	TMatrix  operator-(const TMatrix& mt);
	TMatrix operator*(const TMatrix& mt);
	friend istream& operator>>(istream& in, TMatrix& mt)
	{
		cout << "Enter elements of matrix vectors" << endl;
		for (int i = 0; i < mt.size; i++)
				in >> mt.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;

	}
};
```

Класс наследуется (тип наследования pubic) от класса **TVector <TVector<T>>**

Назначение: представление матрицы как вектор векторов.

Поля:

Size – размерность матрицы.

Start_Index – индекс первого необходимого элемента.

pVec – память для представления элементов матрицы.

Методы:

- **TMatrix (int mn = 10);**

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **mn** – длина вектора (по умолчанию 10).

- **`TMatrix (const TMatrix& m);`**

  Назначение: конструктор копирования.

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

- **`TMatrix (const TVector <TVector<T>>& m);`**

  Назначение: конструктор преобразования типов.

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

  Операторы:

- **`const TMatrix operator= (const TMatrix& m);`**

  Назначение: оператор присваивания.

  Входные параметры: **`m`** – экземпляр класса, который присваиваем.

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

- **`int operator== (const TMatrix& m) const;`**

  Назначение: оператор сравнения.

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

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

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

  Назначение: оператор сравнения.

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

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

- **`TMatrix operator+(const TMatrix& m);`**

  Назначение: оператор сложения матриц.

  Входные параметры: **`m`** – матрица, которую суммируем.

  Выходные параметры: экземпляр класса, равный сумме двух матриц.

- **`TMatrix operator-(const TMatrix& m);`**

  Назначение: оператор вычитания матриц.

  Входные параметры: **`m`** – матрица, которую вычитаем.

  Выходные параметры: экземпляр класса, равный разности двух матриц.

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

Назначение: оператор умножения матриц.

Входные параметры: **`m`** – матрица, которую умножаем.

Выходные параметры: экземпляр класса, равный произведению двух матриц.

- **`template<typename T> friend std::istream& operator>>(std::istream& istr, TMatrix<T>& v);`**

  Назначение: оператор ввода матрицы.

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

  Выходные параметры: поток ввода.

- **`template<typename T> friend std::ostream& operator<<(std::ostream& ostr, const TMatrix<T>& v);`**

  Назначение: оператор вывода матрицы.

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

  Выходные параметры: поток вывода.

Заключение

В результате данной лабораторной работы был разработан шаблонный класс вектор, который поддерживает следующие операции: добавление элемента, удаление элемента, доступ к элементу по индексу и другие. На его основе был мы разработан шаблонный класс для реализации верхнетреугольной матрицы, который также поддерживает различные операции. В целом можно сказать, что проведенный анализ результатов показал, что использование векторов и матриц может быть очень полезным в решении определенных задач.

Литература

1. Определитель верхнетреугольной матрицы [http://elisey-ka.ru/algem/66.htm].
2. Треугольные, транспонированные и симметричные матрицы [https://portal.tpu.ru/SHARED/k/KONVAL/Sites/Russian_sites/1/10.htm].
3. Лекция «Вектора и матрицы» Сысоев А. В. [https://cloud.unn.ru/s/FkYBW5rJLDCgBmJ].

Приложения

Приложение А. Реализация класса TVector

```
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&           mt)           :
TVector<TVector<ValueType>>(mt) { }

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

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

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

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

template <typename ValueType>
TMatrix<ValueType>   TMatrix<ValueType>::operator+(const   TMatrix<ValueType>&
mt)
{
      if (size != mt.size) {
            throw ("Matrices must have the same size ");
      }
      TMatrix tmp(*this);
      for (int i = 0; i <size; ++i) {
            tmp.pVector[i] = tmp.pVector[i] + mt.pVector[i];
      }
      return tmp;
}

template <typename ValueType>
TMatrix<ValueType>   TMatrix<ValueType>::operator-(const   TMatrix<ValueType>&
mt)
{
      if (size != mt.size) {
            throw ("Matrices must have the same size ");
      }
      TMatrix tmp(*this);
      for (int i = 0; i <size; ++i) {
            tmp.pVector[i] = tmp.pVector[i] - mt.pVector[i];
```

```
	}
	return tmp;
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator*(const TMatrix& m) {
	if (size != m.size)
		throw ("Matrices must have the same size ");
	int size =GetSize();
	TMatrix<ValueType> tmp(size);
	for (int k = 0; k < size; k++) {
		for (int j = k; j < size; j++) {
			ValueType sum = 0;
			for (int r = k; r <= j; r++) {
				sum += this->pVector[k][r - k] * m.pVector[r][j - r];
			}
			tmp.pVector[k][j - k] = sum;
		}
	}
	return tmp;
}
```

Приложение Б. 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&		mt)		:
TVector<TVector<ValueType>>(mt) { }

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

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

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

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

template <typename ValueType>
TMatrix<ValueType>	TMatrix<ValueType>::operator+(const	TMatrix<ValueType>&
mt)
{
	if (size != mt.size) {
```

```
            throw ("Matrices must have the same size ");
      }
      TMatrix tmp(*this);
      for (int i = 0; i <size; ++i) {
            tmp.pVector[i] = tmp.pVector[i] + mt.pVector[i];
      }
      return tmp;
}

template <typename ValueType>
TMatrix<ValueType>   TMatrix<ValueType>::operator-(const   TMatrix<ValueType>&
mt)
{
      if (size != mt.size) {
            throw ("Matrices must have the same size ");
      }
      TMatrix tmp(*this);
      for (int i = 0; i <size; ++i) {
            tmp.pVector[i] = tmp.pVector[i] - mt.pVector[i];
      }
      return tmp;
}

template <typename ValueType>
TMatrix<ValueType> TMatrix<ValueType>::operator*(const TMatrix& m) {
      if (size != m.size)
            throw ("Matrices must have the same size ");
      int size =GetSize();
      TMatrix<ValueType> tmp(size);
      for (int k = 0; k < size; k++) {
            for (int j = k; j < size; j++) {
                  ValueType sum = 0;
                  for (int r = k; r <= j; r++) {
                        sum += this->pVector[k][r - k] * m.pVector[r][j - r];
                  }
                  tmp.pVector[k][j - k] = sum;
            }
      }
      return tmp;
}
```