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....4
1 Постановка задачи....5
2 Руководство пользователя....6
2.1 Приложение для демонстрации работы векторов....6
2.2 Приложение для демонстрации работы матриц....8
3 Руководство программиста....12
3.1 Описание алгоритмов....12
3.1.1 Векторы....12
3.1.2 Матрицы....13
3.2 Описание программной реализации....18
3.2.1 Описание класса TVector....18
3.2.2 Описание класса TMatrix....22
Заключение....26
Литература....27
Приложения....28
Приложение А. Реализация класса TVector....28
Приложение Б. Реализация класса TMatrix....32

Введение

.Ранее уже рассматривался шаблонный класс TVector. Теперь используем его в качестве интерфейса для создания нового шаблонного класса для работы с матрицами. В данной лабораторной работе рассматриваются только матрицы специального вида – верхне-треугольные. Поэтому реализация класса матриц будет несколько отличаться от реализации матриц общего вида.

1 Постановка задачи

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

Задачи:

1. Разработать класс TVector. Класс должен поддерживать следующие операции:сложение,вычитание,копирование векторов, равенство,неравенство.
2. Разработать класс TMatrix. Класс 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_TBitField.exe. В результате появится окно, показанное на рисунке (рис. 1)

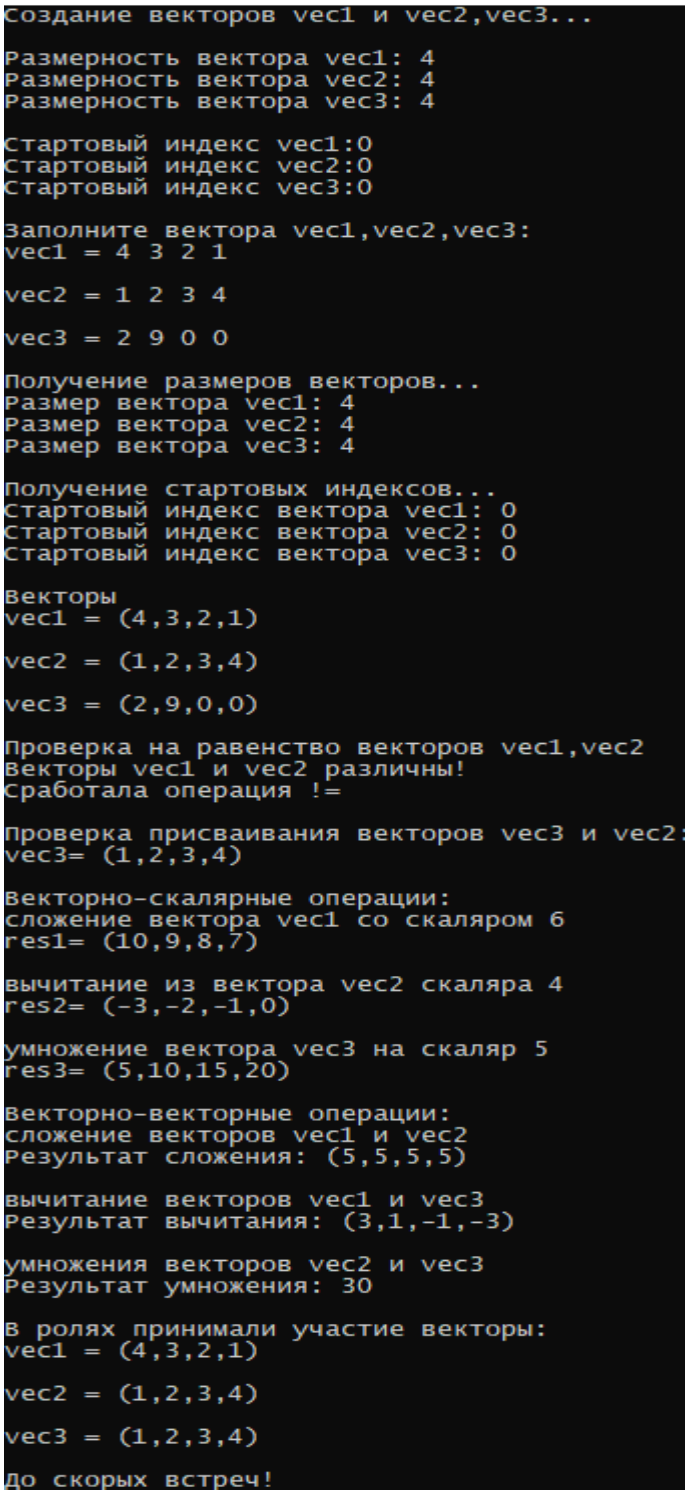


Рис. 1. Основное окно программы

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

```
Создание векторов vec1 и vec2,vec3...
```

Рис. 2. Создание векторов

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

```
vec1 = 4 3 2 1

vec2 = 1 2 3 4

vec3 = 2 9 0 0

Получение размеров векторов...
Размер вектора vec1: 4
Размер вектора vec2: 4
Размер вектора vec3: 4

Получение стартовых индексов...
Стартовый индекс вектора vec1: 0
Стартовый индекс вектора vec2: 0
Стартовый индекс вектора vec3: 0

Векторы
vec1 = (4,3,2,1)

vec2 = (1,2,3,4)

vec3 = (2,9,0,0)
```

Рис. 3. Заполнение векторов и вывод

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

```
Проверка на равенство векторов vec1,vec2
Векторы vec1 и vec2 различны!
Сработала операция !=
```

Рис. 4. Проверка на равенство векторов

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

```
Проверка присваивания векторов vec3 и vec2:
vec3= (1,2,3,4)
```

Рис. 5. присваивание векторов

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

```
Векторно-скалярные операции:
сложение вектора vec1 со скаляром 6
vec1= (10,9,8,7)

вычитание из вектора vec2 скаляра 4
vec2= (-3,-2,-1,0)

умножение вектора vec3 на скаляр 5
vec3= (5,10,15,20)
```

Рис. 6. Векторно-скалярные операции с векторами

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

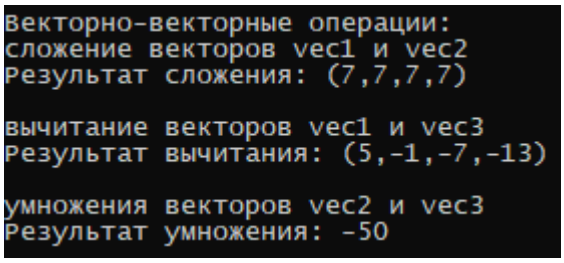


Рис. 7. Векторно-векторные операции с векторами

8. Для сравнения результатов векторы выводятся на экран (**Error! Reference source not found.**)

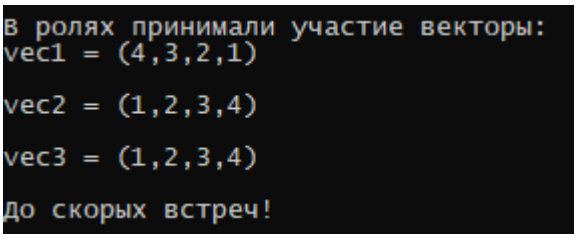


Рис. 8. Вывод

2.2 Приложение для демонстрации работы матриц

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

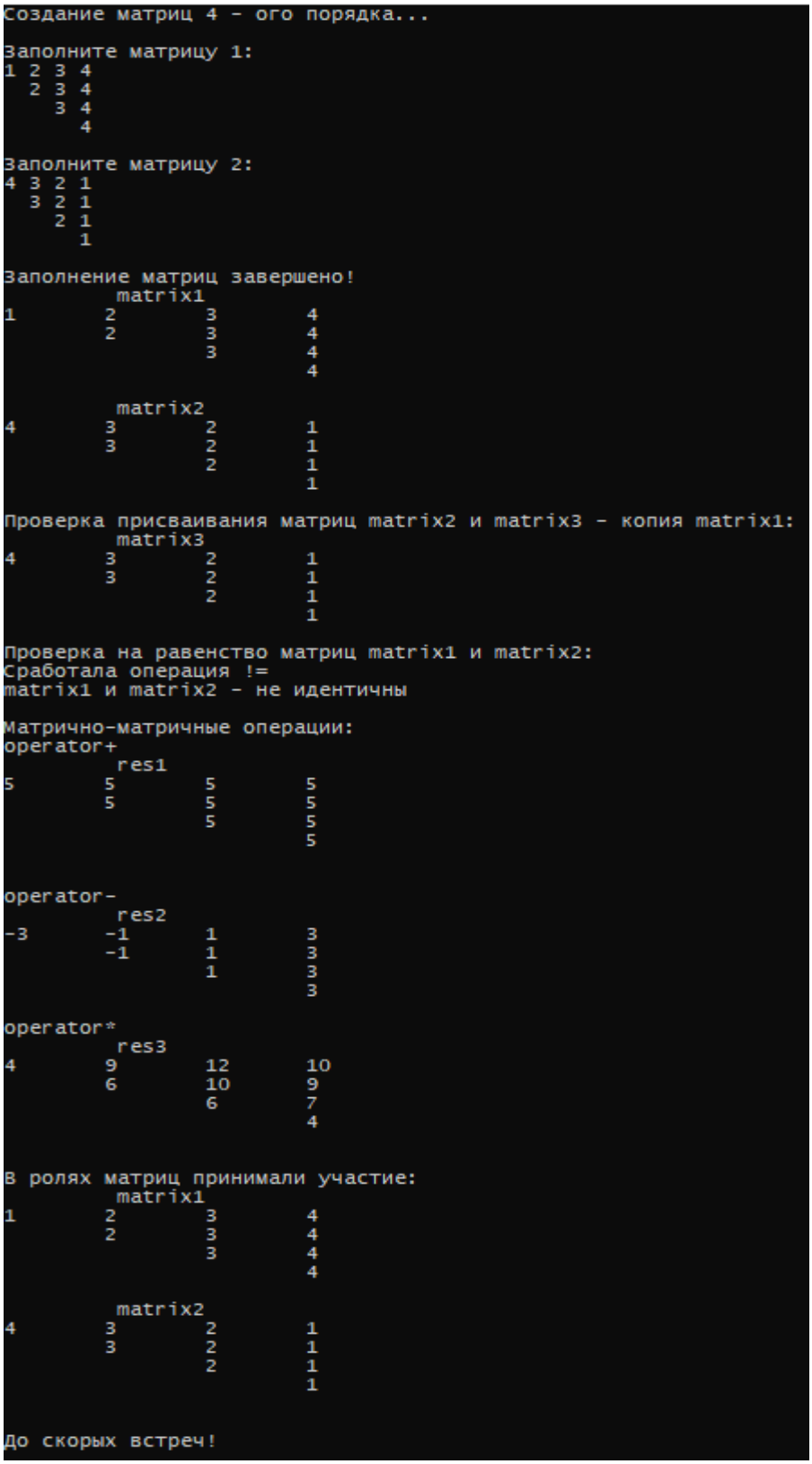


Рис.1 Окно основной программы

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

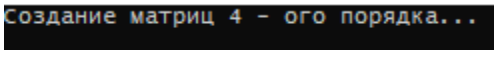


Рис.2. Создание матриц

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

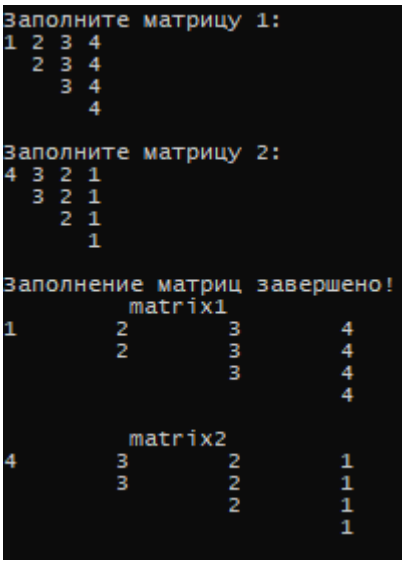


Рис.3 Заполнение матриц и вывод

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

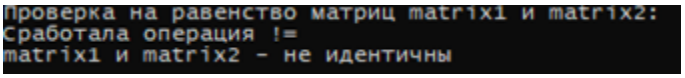


Рис.4 Сравнение матриц

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

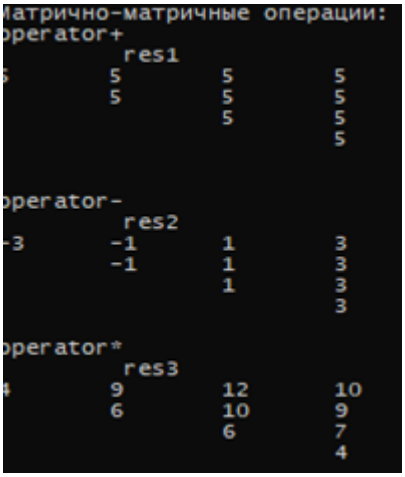


Рис.5 работа с матрицами

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

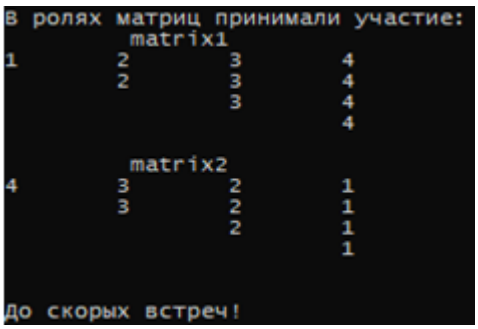

В ролях матриц принимали участие:
matrix1
1 2 3 4
2 3 4
3 4
4
matrix2
4 3 2 1
3 2 1
2 1
1
До скорых встреч!

Рис.6 Вывод

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Векторы

Шаблонный класс TVector хранится как динамический одномерный массив. Элементы имеют тип Type(параметр шаблона). У каждого вектора есть свой стартовый индекс, начиная с которого идёт выделение памяти под массив элементов, и размер.

Рассмотрим базовые операции для работы с векторами.

1. Операция квадратные скобки. Чтобы получить доступ к компоненте вектора,необходимо перегрузить данную операцию для осуществления основных операций с векторами. В качестве примера возьмём вектор A со стартовым индексом 1 и размером 4.

Index	0	1	2	3
Vector A	1	2	3	4

Известно.что выделенная память расположена от start_index до pos-start_index. Это нужно затем, чтобы не хранить нулевые элементы матрицы, расположенные ниже главной диагонали.

```
TVector<int> vector_A(4,1);
Vector_A[2];
```

В результате будет получен доступ к элементу с индексом 1.

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 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

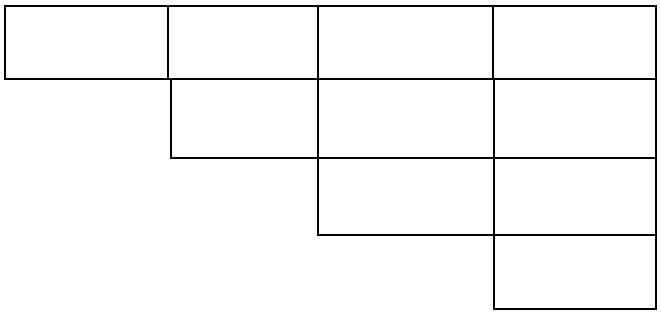
Для того чтобы поддерживать эффективное хранение матриц, шаблонный класс TMatrix наследуется от класса TVector как вектор векторов. Другими словами, мы имеем массив массивов,в котором каждый элемент является объектом класса TVector. Это позволяет описать работу матриц наиболее просто и понятно. Аналогом

матриц может,например,послужить двумерный массив, с которым уже работали ранее.

<TVector> - параметр шаблона класса TVector

<Type> - параметр шаблона класса TMatrix

Рассматриваемый тип матриц – верхне-треугольные



Перейдём к рассмотрению базовых операций над матрицами.

1. Операция квадратные скобки. Для доступа к элементамм матрицы необходимо дважды использовать данную операцию. В первый раз мы обращаемся к i-той строке(i-тый вектор вектора векторов),а во второй раз – к j-ому столбцу(j-тая компонента i-того вектора). Приведём показательный пример на матрице 4-ого порядка

3	2	3	4
	2	1	4
		3	3
			4

```
TMatrix<double> matr1(4);
for (int i = 0; i < matr1.GetSize(); i++) {
 for (int k = matr1.GetStart() + i; k < matr1.GetSize(); k++) {
       matr1[i][k] = k;
 }
}
```

Приведённый выше код демонстрирует работу операции квадратные скобки для матриц. Сначала указывается i-тый вектор,а потом - k-тая компонента i-того вектора. Есть сходство с двумерным массивом.

2. Операция сложения. Чтобы сложить две матрицы, необходимо сложить компоненты соответствующих векторов-строк.

Матрица 1

3	2	3	4
	2	1	4
		3	3
			4

Матрица 2

3	2	0	4
	2	1	4
		3	3
			2

Результирующая матрица

6	4	3	8
	4	2	8
		6	6
			6

Каждый элемент одной матрицы соответственно складывается с элементом другой матрицы.

3. Операция вычитания. Чтобы найти разность матриц, нужно найти разности соответствующих векторов-строк.

Матрица 1

3	2	3	4
	2	1	4
		3	3
			4

Матрица 2

4	2	0	2
	6	1	8
		7	3
			3

Результирующая матрица

-1	0	3	2
	-4	0	-4
		0	0
			1

4. Умножение матриц специального вида. Чтобы умножить две треугольные матрицы, нужно соотнести стартовые индексы одного вектора-строки матрицы с вектором-строки другой матрицы, потому что не учитываем нулевые элементы матриц.

Общий принцип:
Определим функцию, которая каждую строку одной матрицы сопоставляет с некоторой подматрицей порядка либо равного другой матрице,либо меньше. Каждой строке ставится в соответствие нижняя граница – стартовый индекс(фиксированный) s_i-вектора строки и верхняя граница - стартовый индекс строки s_{i+j},где j = 1,2,3,…

$F(s_i) \rightarrow B_m$ принадлежит A^{m*n} и

Теперь чтобы получить результирующую строку,нужно сопоставить число пар произведений элементов строки s_i матрицы A согласно выбранным границам.

Матрица 1

Первая строка сопоставляется с подматрицей ранга 4.

3	2	3	4
	2	1	4
		3	3
			4

Матрица 2

4	2	0	2
	6	1	8
		7	3
			3

Результирующая матрица

12	18	23	43
	12	9	31
		21	18
			12

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:

Конструктор инициализатор

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

Входные параметры: size – размер вектора, start_index – стартовый индекс

Выходные параметры: отсутствуют

Конструктор копирования

Назначение: копирование векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: отсутствуют

деструктор

Назначение: освобождение памяти vector

Входные параметры: отсутствуют

Выходные параметры: отсутствуют

GetSize

Назначение: получение размера вектора

Входные параметры: отсутствуют

Выходные параметры: size – размер вектора

GetStart

Назначение: получение стартового индекса

Входные параметры: отсутствуют

Выходные параметры: start_index – стартовый индекс

Operator[]

Назначение: получение элемента памяти vector

Входные параметры: index – номер элемента

Выходные параметры: элемент с номером index-start_index

Operator==

Назначение: сравнение на равенство векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: целое число – 0 или 1

Operator!=

Назначение: сравнение на равенство векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: целое число – 0 или 1

Operator=

Назначение: присваивание полей

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: ссылка на объект себя

Operator+

Назначение: сложить вектор со скаляром

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат сложения

Operator-

Назначение: вычесть из вектора скаляр

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат вычитания

Operator*

Назначение: умножить вектор на скаляр

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат умножения

Operator+

Назначение:

Входные параметры:

Выходные параметры:

Operator+

Назначение: сложить векторы

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат сложения

Operator-

Назначение: вычесть один вектор из другого

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат вычитания

Operator+

Назначение: умножить один вектор на другой

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат умножения

Operator>>

Назначение: ввод элементов вектора

Входные параметры: istr-ссылка на стандартный поток ввода,obj – неконстантная ссылка на объект класса TVector

Выходные параметры: istr- ссылка на стандартный поток ввода

Operator<<

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;
}

};
```

<TVector> - параметр шаблона класса TVector

<Type> - параметр шаблона класса TMatrix

Методы:

Конструктор инициализатор

Назначение: выделение памяти под каждый вектор вектора векторов

Входные параметры: mp – размерность матрицы

Выходные параметры: отсутствуют

Конструктор копирования

Назначение: копирование матриц

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: отсутствуют

Конструктор преобразования типа

Назначение: преобразует вектор векторов в матрицу

Входные параметры: TVector<TVector<Type>& v – константная ссылка на объект класса TVector

Выходные параметры: отсутствуют

Operator=

Назначение: присваивание матриц

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: ссылка объект себя

Operator==

Назначение: проверка матриц на равенство

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: целое число – 0 или 1

Operator!=

Назначение: проверка матриц на неравенство

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: целое число – 0 или 1

Operator+

Назначение: сложить матрицы

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: новый объект класса TMatrix как результат сложения

Operator-

Назначение: вычесть из одной матрицы другую

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: константная ссылка на объект класса TMatrix как результат вычитания

Operator*

Назначение: умножить матрицу на матрицу

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: константная ссылка на объект класса TMatrixкак результат умножения

Operator<<

Назначение: вывод матриц

Входные параметры: ostr-ссылка на стандартный поток ввода,obj – константная ссылка на объект класса TMatrix

Выходные параметры:ostr- ссылка на стандартный поток вывода

Operator>>

Назначение: ввод матриц

Входные параметры: istr-ссылка на стандартный поток ввода,obj – неконстантная ссылка на объект класса TMatrix

Выходные параметры: istr – ссылка на стандартный поток ввода

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

Н.В Гредасов, Линейная Алгебра – Издательство о Уральского университета. Редакционно-издательский отдел ИПЦ УрФУ 620049, 92 страницы (с.6)

Алексей Померанцев, Векторы и матрицы – Российское Хемометрическое общество,33 раздела(р.6)

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;
}
```