

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА

на тему:

«Векторы и матрицы»

Выполнил(а): студент(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	9
3 Руководство программиста	11
3.1 Описание алгоритмов	11
3.1.1 Векторы	11
3.1.2 Матрицы.....	16
3.2 Описание программной реализации	19
3.2.1 Описание класса TVector.....	19
3.2.2 Описание класса TMatrix.....	24
Заключение	27
Литература	28
Приложения	29
Приложение А. Реализация класса TVector	29
Приложение Б. Реализация класса TMatrix	35

Введение

Ранее мы уже сталкивались с шаблонным классом `TVector`, который хранил в себе компоненты вектора разного фундаментального типа данных и переменный размер. В данной лабораторной работе мы описываем реализацию класса шаблонного класса `TVector` с целью реализации класса матриц `TMatrix`, наследуемый от класса `TVector`, принимающий в качестве параметра шаблона самого себя. Таким образом, матрица – это класс вектора векторов как интерфейс для работы с матрицами как общего, так и специального вида. В лабораторной работе рассматриваются матрица специального вида – верхне-треугольные.

1 Постановка задачи

Цель – реализовать шаблонные классы: TVector и TMatrix

Задачи:

1. Класс для работы с матрицами должен поддерживать эффективное хранение данных.
2. Написать следующие операции для работы с векторами: сложение, вычитание, умножение, копирование векторов, сравнение(==, !=), ввод/вывод.
3. Добавить вспомогательные операции: [], получение стартового индекса и размера вектора, сложение, вычитание, умножение с элементом.
4. Написать следующие операции для работы с матрицами: сложение, вычитание, умножение матриц, копирование матриц, сравнение(==, !=), ввод/вывод.

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)

```
Создание векторов vec1 и vec2, vec3...
Размерность вектора vec1: 4
Размерность вектора vec2: 4
Размерность вектора vec3: 4

Стартовый индекс vec1: 0
Стартовый индекс vec2: 0
Стартовый индекс vec3: 0

Заполните вектора vec1, vec2, vec3:
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)

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

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

Векторно-скалярные операции:
Сложение вектора vec1 со скаляром 6
res1 = (10,9,8,7)

Вычитание из вектора vec2 скаляра 4
res2 = (-3,-2,-1,0)

Умножение вектора vec3 на скаляр 5
res3 = (5,10,15,20)

Векторно-векторные операции:
Сложение векторов vec1 и vec2
Результат сложения: (5,5,5,5)

Вычитание векторов vec1 и vec3
Результат вычитания: (3,1,-1,-3)

Умножения векторов vec2 и vec3
Результат умножения: 30

В ролях принимали участие векторы:
vec1 = (4,3,2,1)
vec2 = (1,2,3,4)
vec3 = (1,2,3,4)

До скорых встреч!
```

Рис. 1. Основное окно программы

2. На первом шаге создаются 3 вектора(рис.2)

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

Рис.2 Создание векторов

3. На следующем шаге заполняются векторы vec1,vec2,vec3 и выводятся их данные на экран(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 данных

4. Далее идёт 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. На 5 шаге присваиваются значения вектора vec2 вектору vec3(рис.5)

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

Рис.5 Проверка присваивания векторов

6. На данном этапе выполняются векторно-скалярные операции с векторами(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. После проводятся векторно-векторные операции(рис.7)

```
Векторно-векторные операции:  
сложение векторов vec1 и vec2  
Результат сложения: (7,7,7,7)  
  
вычитание векторов vec1 и vec3  
Результат вычитания: (5,-1,-7,-13)  
  
умножения векторов vec2 и vec3  
Результат умножения: -50
```

Рис.7 Векторно-векторные операции с векторами

8. На завершающем этапе выводятся на экран векторы для сравнения результатов(рис.8)

```
В ролях принимали участие векторы:  
vec1 = (4,3,2,1)  
  
vec2 = (1,2,3,4)  
  
vec3 = (1,2,3,4)  
  
До скорых встреч!
```

Рис.8 Текущие векторы

2.2 Приложение для демонстрации работы матриц

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

```
Создание матриц 4 - ого порядка...
Заполните матрицу 1:
1 2 3 4
  2 3 4
    3 4
      4

Заполните матрицу 2:
4 3 2 1
  3 2 1
    2 1
      1

Заполнение матриц завершено!
matrix1
1      2      3      4
      2      3      4
          3      4
              4

matrix2
4      3      2      1
  3      2      1
    2      1
      1

проверка присваивания матриц matrix2 и matrix3 - копия matrix1:
matrix3
4      3      2      1
  3      2      1
    2      1
      1

Проверка на равенство матриц matrix1 и matrix2:
Сработала операция !=
matrix1 и matrix2 - не идентичны

Матрично-матричные операции:
operator+
res1
5      5      5      5
      5      5      5
          5      5
              5

operator-
res2
-3     -1      1      3
      -1      1      3
          1      3
              3

operator*
res3
4      9      12     10
      6      10      9
          6      7
              4

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.1 Окно основной программы

2. На первом шаге создаются матрицы указанного порядка (рис.2)

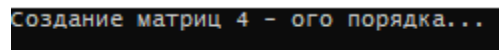
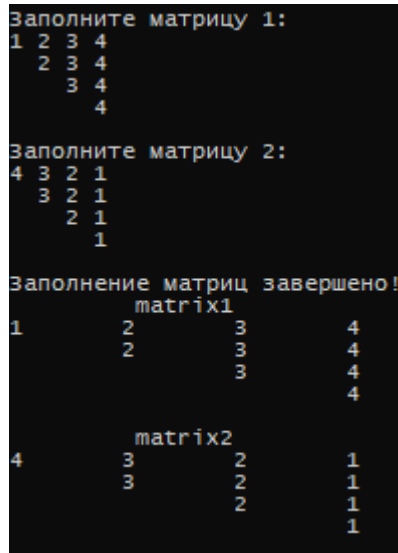


Рис.2. Процесс создания

3. Далее матрицы заполняются значениями того типа, который был указан до этапа создания матриц, и результаты ввода выводятся на экран (рис.3)



```
Заполните матрицу 1:
1 2 3 4
 2 3 4
   3 4
    4

Заполните матрицу 2:
4 3 2 1
 3 2 1
   2 1
    1

Заполнение матриц завершено!
matrix1
1      2      3      4
      2      3      4
        3      4
          4

matrix2
4      3      2      1
 3      2      1
   2      1
    1
```

Рис.3 Заполнение матриц и вывод на экран

4. На представленном рисунке можно увидеть операции сравнения матриц(рис.4)

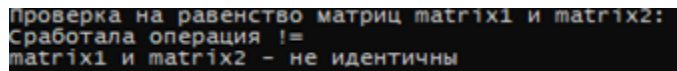
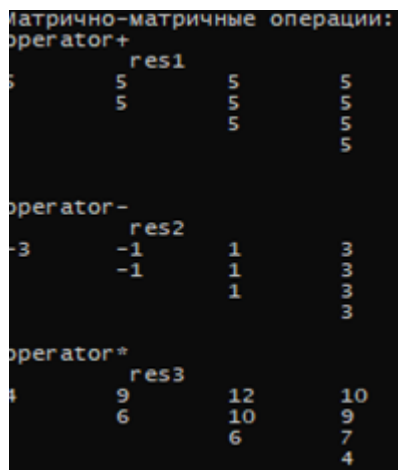


Рис.4 Сравнение матриц

5. На рис.5 приведены матрично-матричные операции(рис.5)



```
Матрично-матричные операции:
operator+
res1
5      5      5      5
 5      5      5
   5      5
    5

operator-
res2
-3      1      3
 -1      1      3
 -1      1      3
      1      3

operator*
res3
9      12     10
 6      10      9
      6      7
        4
```

Рис.5 работа с матрицами

6. В завершение выводятся матрицы, которые принимали участие в работе(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 матриц

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Векторы

Шаблонный класс вектор представляет из себя C/X – динамический одномерный массив с элементами типа Type(параметр шаблона). У каждого вектора есть свой стартовый индекс, с которого выделяется память для хранения элементов, и размер. Используется как интерфейс для работы с матрицами.

Методы:

Конструктор с параметрами по умолчанию.

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

Выходные параметры: отсутствуют

Алгоритм:

1. Выполняются проверки на отрицательность размера и максимально допустимое значение и на отрицательность стартового индекса.
2. Поля size и start_index инициализируются значениями переданных параметров.
3. Выделяется память в количестве size под хранение элементов вектора.

Конструктор копирования

Входные данные: константная ссылка на объект типа TVector

Выходные данные: отсутствуют

Алгоритм:

1. Поля `size` и `start_index` инициализируются значениями полей переданного объекта.
2. Выделяется память в количестве `size` под хранение элементов вектора
3. В цикле `for` творится копирование значение элементов из поля `vector` переданного объекта в поле `vector` текущего объекта типа `TVector`.

Деструктор

Входные данные: отсутствуют

Выходные данные: отсутствуют

Алгоритм:

1. Освобождает память из-под массива `vector`.

`GetSize`

Входные параметры: отсутствуют

Выходные параметры: `size` – значение поля `size`

Алгоритм:

Возврат значения поля `size`

`GetStart`

Входные параметры: отсутствуют

Выходные параметры: `start_index` – значение поля `start_index`

Алгоритм:

Возврат значения поля `start_index`

Оператор `==`

Входные параметры: константная ссылка на объект типа `TVector`

Выходные параметры: целое число – 0 или 1

Алгоритм:

1. Проверка на равенство размеров. В случае неравенства возвращает `false`.
2. Проверка на стартовый индекс. В случае неравенства возвращает `false`.
3. Поэлементное сравнение компонент вектора с элементами вектора переданного объекта. В случае несовпадения хотя бы с одним возвращается `false`.

4. Возвращается true, если все условия не выполняются

Operator !=

Входные параметры: константная ссылка на объект типа TVector

Выходные параметры: целое число – 0 или 1

Алгоритм:

Возвращается инвертированный результат операции ==

Operator=

Входные параметры: константная ссылка на объект типа TVector

Выходные параметры: ссылка на объект себя

Алгоритм:

1. Проверка стартовых индексов. В случае неравенства переприсваивание поля start_index.
2. Проверка на равенство размеров. В случае неравенства удаляется старая память, поле size инициализируется новым значением, память перевыделяется в количестве size.
3. Поэлементное присваивание компонент вектора текущего объекта и элементами вектора переданного объекта.
4. Возвращается *this.

Operator[]

Входные данные: index – номер элемента

Выходные данные: элемент вектора

Алгоритм:

1. Проверка индекса на отрицательность и на обращение к мнимой части (невыведенная память). Бросается исключение в случае неуспешной проверки.
2. Возвращается номер элемента index-start_index, так как не учитываем мнимую часть.

Operator+

Входные параметры: константная ссылка на объект типа TVector

Выходные параметры: новый объект класса типа TVector

Алгоритм:

1. Проверка на стартовые индексы. Бросается исключение в случае неравенства.
2. Проверка на равенство размеров. Бросается исключение в случае неравенства.
3. Создаётся копия текущего объекта под результат.
4. Выполняется поэлементное сложение элементов вектора текущего объекта с элементами переданного объекта типа TVector.
5. Возвращается результирующий объект типа TVector.

Operator-

Входные параметры: константная ссылка на объект типа TVector

Выходные параметры: новый объект класса типа TVector

Алгоритм:

1. Проверка на стартовые индексы. Бросается исключение в случае неравенства.
2. Проверка на равенство размеров. Бросается исключение в случае неравенства.
3. Создаётся копия текущего объекта под результат.
4. Выполняется поэлементная разность элементов вектора текущего объекта с элементами переданного объекта типа TVector.
5. Возвращается результирующий объект типа TVector.

Operator*

Входные параметры: константная ссылка на объект типа TVector

Выходные параметры: новый объект класса типа TVector

Алгоритм:

1. Проверка на стартовые индексы. Бросается исключение в случае неравенства.
2. Проверка на равенство размеров. Бросается исключение в случае неравенства.
3. Создаётся копия текущего объекта под результат.
4. Выполняется поэлементное сложение пар произведений соответствующих элементов вектора текущего объекта с элементами переданного объекта типа TVector.

5. Возвращается результирующий объект типа TVector.

Operator>>

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

Выходные параметры: istr- ссылка на стандартный поток ввода

Алгоритм:

1. Создается внешний цикл for для прохождения по элементам вектора
2. Внутренний цикл for начинает отсчёт от стартового индекса до размера вектора.
3. Выполняется заполнение на указанном промежутке.
4. Возвращается ссылка на стандартный поток ввода.

Operator<<

Входные параметры: ostr-ссылка на стандартный поток ввода,obj – константная ссылка на объект типа TVector

Выходные параметры: ostr- ссылка на стандартный поток вывода

Алгоритм:

1. Создается внешний цикл for для прохождения по элементам вектора
2. Внутренний цикл for начинает отсчёт от стартового индекса до размера вектора.
3. Выполняется вывод на указанном промежутке.
4. Возвращается ссылка на стандартный поток вывода.

3.1.2 Матрицы

Шаблонный класс TMatrix наследуется от класса TVector как вектор векторов(public наследование). Класс матриц полностью содержит интерфейс класса TVector и использует его для осуществления матричных операций. В целом, реализация TMatrix основывается на использовании готовых методов TVector.

<TVector> - параметр шаблона класса TVector

<Type> - параметр шаблона класса TMatrix

Методы:

Конструктор инициализатор

Входные параметры: mр – размерность матрица

Выходные параметры: отсутствуют

Алгоритм:

1. Вызов конструктора инициализатора базового класса.
2. В цикле for для каждого вектора vector[i] вектора векторов вызывается конструктор инициализатор базового класса TVector

Конструктор копирования

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: отсутствуют

Алгоритм:

Вызывается конструктор копирования базового класса TVector

Конструктор преобразования типа

Входные параметры: константная ссылка на объект TVector<TVector<Type>>

Выходные параметры: отсутствуют

Алгоритм:

Вызов конструктора копирования базового класса TVector

Operator=

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: ссылка на объект себя

Алгоритм:

Вызов оператора присваивания базового класса TVector

Operator==

Входные данные: константная ссылка на объект класса TMatrix

Выходные данные: целое число – 0 или 1

Алгоритм:

Вызов оператора == базового класса TVector

Operator!=

Входные данные: константная ссылка на объект класса TMatrix

Выходные данные: целое число – 0 или 1

Алгоритм:

Вызов оператора != базового класса TVector

Operator+

Входные данные: константная ссылка на объект класса TMatrix

Выходные данные: новый объект класса TMatrix

Алгоритм:

Вызов оператора + базового класса TVector

Operator-

Входные данные: константная ссылка на объект класса TMatrix

Выходные данные: новый объект класса TMatrix

Алгоритм:

Вызов оператора - базового класса TVector

Operator*

Входные данные: константная ссылка на объект класса TMatrix

Выходные данные: новый объект класса TMatrix

Алгоритм:

1. Выполняется проверка на равенство размерностей матриц. В случае неравенства бросается исключение.
2. Выполняется проверка на равенство стартовых индексов матриц. В случае неравенства бросается исключение.
3. Создаётся вектор векторов с вызовом конструктора инициализатор, принимающего размерность матрицы своего класса TMatrix.
4. Создаётся внешний цикл for для прохода по элементам вектора векторов.
5. Создаётся внутренний цикл for для прохода по значениям элементов вектора векторов.
6. Цикл for внутри внутреннего цикла будет отвечать за проход по столбцам матрицы.
7. По формуле вычисляется каждый результирующий элемент результирующей матрицы(вектора-векторов) в результате умножения соответствующих элементов матриц.
8. Возвращается конструктор преобразования типа, который принимает вектор-векторов как результат умножения матриц.

Operator>>

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

Выходные данные: istr- ссылка на стандартный поток ввода

Алгоритм:

1. Создаётся внешний цикл for для прохода по элементам вектора векторов
2. Создаётся внутренний цикл for для заполнения элементов вектора векторов значениями(ЗАПОЛНЯЕТСЯ ТОЛЬКО ТА ЧАСТЬ ВЕКТОРА, ПОД КОТОРУЮ БЫЛА ВЕДЕЛЕНА ПАМЯТЬ)
3. Возвращается ссылка на стандартный поток ввода.

Operator<<

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

Выходные данные: ostr- ссылка на стандартный поток вывода

Алгоритм:

1. Создаётся внешний цикл for для прохода по элементам вектора векторов
2. Создаётся первый внутренний цикл for для вывода символа табуляции.
3. Второй внутренний цикл for отвечает за вывод значений элементов вектора векторов(нули под главной диагональю не выводятся)
4. Возвращается ссылка на стандартный поток вывода.

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: mpr – 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 TMatrix как 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 для работы с векторами и матрицами. В ходе лабораторной работы была выведена формула для умножения матриц специального вида – верхне-треугольные,а также были 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

```

//#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;
};

template <class Type>
TVector<Type>::TVector(int size_, int start_index_) {
    if (size_ <= 0 || size_ > INT_MAX) {
        throw Exceptions<int>(WRONG_SIZE, size_);
    }
    if (start_index_ < 0) {
        throw Exceptions<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 Exceptions<int>(WRONG_INDEX, index);
    }
    if (index < start_index) {
        throw Exceptions<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 Exceptions<int>(WRONG_INDEX, start_index);
    if (size != obj.size) {
        throw Exceptions<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 Exceptions<int>(WRONG_INDEX, start_index);
    if (size != obj.size) {
        throw Exceptions<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 Exceptions<int>(WRONG_INDEX, start_index);
    if (size != obj.size) {
        throw Exceptions<int>(WRONG_SIZE, size);
    }
    Type result=0;
    for (int i = 0; i < size; i++) {
        result = result + (vector[i] * obj.vector[i]);
    }
    return result;
}

```



```

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 полях 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

```
//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 << " ";
        }
        //пропуск нулевых элементов
        for (int k = obj.GetStart()+i; k < obj.GetSize(); k++) {
            ostr << obj.vector[i][k] << " ";
        }
        ostr << endl;
    }
    return ostr;
}

};

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 Exceptions<int>(WRONG_SIZE, size);
    if (start_index != matr.GetStart()) throw Exceptions<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);
}
```

```

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 полях 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;

```