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 3](#_Toc152003377)

[1 Постановка задачи 4](#_Toc152003378)

[2 Руководство пользователя 5](#_Toc152003379)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc152003380)

[2.2 Приложение для демонстрации работы матриц 7](#_Toc152003381)

[3 Руководство программиста 11](#_Toc152003382)

[3.1 Описание алгоритмов 11](#_Toc152003383)

[3.1.1 Векторы 11](#_Toc152003384)

[3.1.2 Матрицы 12](#_Toc152003385)

[3.2 Описание программной реализации 17](#_Toc152003386)

[3.2.1 Описание класса TVector 17](#_Toc152003387)

[3.2.2 Описание класса TMatrix 21](#_Toc152003388)

[Заключение 25](#_Toc152003389)

[Литература 26](#_Toc152003390)

[Приложения 27](#_Toc152003391)

[Приложение А. Реализация класса TVector 27](#_Toc152003392)

[Приложение Б. Реализация класса TMatrix 31](#_Toc152003393)

# Введение

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

# Постановка задачи

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

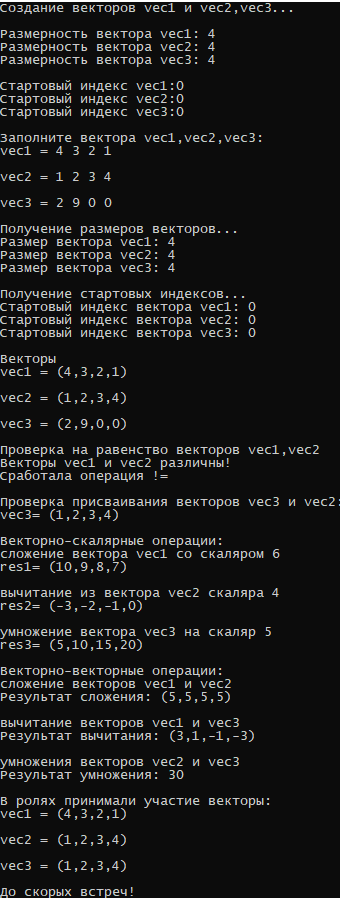
Задачи:

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

# Руководство пользователя

## Приложение для демонстрации работы векторов

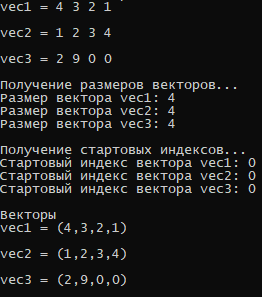
1. Запустите приложение с названием sample\_TVector.exe. В результате появится окно, показанное на рисунке ().



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



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



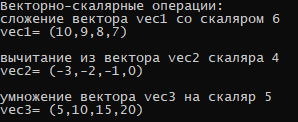
1. Заполнение векторов и вывод
2. На рисунке ниже демонстрируется сравнение векторов vec1 и vec2().



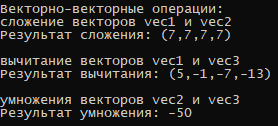
1. Проверка на равенство векторов
2. Далее представлена работа операции присваивания векторов ().



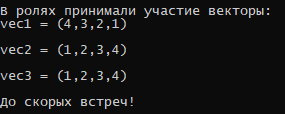
1. присваивание векторов
2. Примеры работы векторно-скалярных операций ().



1. Векторно-скалярные операции с векторами
2. Примеры векторно-векторных операций(рис. 7).



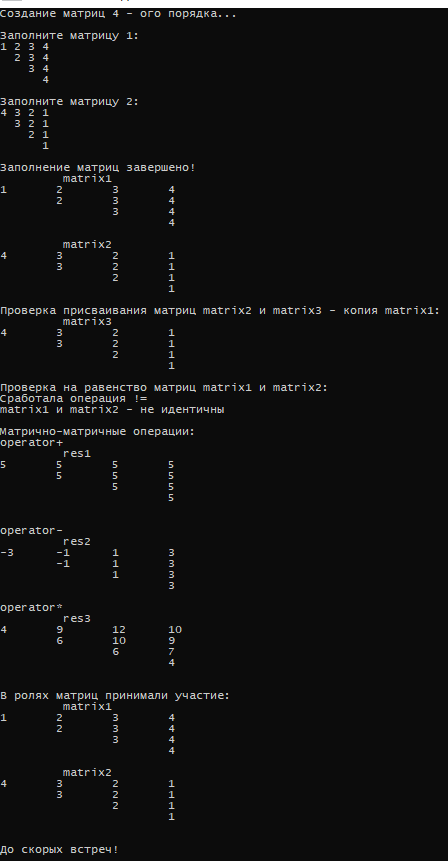
1. Векторно-векторные операции с векторами
2. Для сравнения результатов векторы выводятся на экран ().



1. Вывод

## Приложение для демонстрации работы матриц

1. Запустите приложение с названием sample\_TMatrix.exe. В результате появится окно, показанное ниже ().

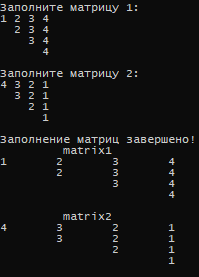


1. Окно основной программы
2. В начале работы программы создаются матрицы указанного порядка

().



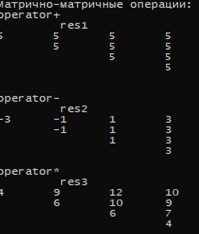
1. Создание матриц
2. На данном этапе пользователь вводит соответствующие значения элементов матриц,и результаты заполнения выводятся на экран ().



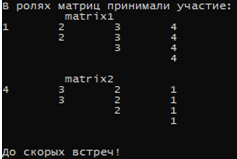
1. Заполнение матриц и вывод
2. На рисунке ниже сравниваются две матрицы – matrix1 и matrix2 ().



1. Сравнение матриц
2. Примеры матрично-матричных операций ().



1. Работа с матрицами
2. Для сравнения результатов матрицы выводятся на консоль ().



1. Вывод

# Руководство программиста

## Описание алгоритмов

### Векторы

Векторы хранятся как динамический одномерный массив, в котором компоненты вектора имеют шаблонный тип. У каждого вектора есть стартовый индекс и размер – количество компонент. Стартовый индекс – это индекс, начиная с которого можно получить доступ к компонентам вектора.

Рассмотрим базовые операции для работы с векторами.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 |
| Vector A | 1 | 2 | 3 | 4 |

Известно, что выделенная память расположена от start\_index до pos-start\_index, где pos – это индекс, по которому мы хотим получить компоненту вектора. Данная стратегия выделения памяти необходима, чтобы не хранить нулевые элементы матрицы, расположенные ниже главной диагонали. Пример со стартовым индексом 0.

Vector<int> A(5);

cout << A[0] << endl;

В результате будет получена компонента вектора с индексом 0.

1. 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 |

1. 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 |

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

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

(A,B) = ∑(Ai\*Bi) = 1\*3 + 2\*0 + 3\*9 + 4\*(-4) = 14

1. 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 |

### Матрицы

Матрица хранится как вектор векторов. Другими словами, как массив массивов. Каждый элемент такого массива – это вектор. Легко заметить, что матрицы полностью основаны на векторах. Поэтому работать с матрицами будем как с массивом векторов, а это напоминает работу с двумерным массивом, в котором доступ к элементу осуществляется через двойные квадратные скобки.

Рассматриваемый тип матриц – верхне-треугольные, где элементы ниже главной диагонали равны 0.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | | 3 | 4 | |
| 5 | 6 | | 7 |
| 8 | | 9 |
| 10 |

Перейдём к рассмотрению базовых операций над матрицами.

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

Matrix<int> A(4);

cout << A[1][2] << endl;

;

В результате буден получен элемент a12,где i=1,j=2.

1. Операция сложения. Каждый элемент одной матрицы складывается с элементом другой матрицы соответственно.

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 |

1. Операция вычитания. Из каждого элемента одной матрицы вычитается элемент другой матрицы соответственно.

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 |

1. Умножение матриц специального вида.

Общий принцип:

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

i,j = 0,size и k = start\_index+I,start\_index+j+1

Пояснение:

Каждой итерации i соответствует результирующая строка элементов новой матрицы.

a(i,k) – это элементы i-той строки k-столбца матрицы 1

b(k,j) – это элементы k-той строки j-того стобца матрицы 2

Первая строка сопоставляется с подматрицей ранга 4.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12 | 18 | | 23 | 43 | |
| 12 | 9 | | 31 |
| 21 | | 18 |
| 12 |

## Описание программной реализации

### Описание класса TVector

template <class Type> class TVector {

protected:

int start\_index;

int size;

Type\* vector;

public:

//#конструкторы и деструктор

TVector(int size\_ = 10, int start\_index\_ = 0);

TVector(const TVector<Type>& obj);

~TVector();

//#свойства вектора

int GetSize() const;

int GetStart() const;

Type& operator[](const int index);

//#сравнение векторов

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

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

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

//#векторно-скалярные операции

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

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

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

//#векторно-векторные операции

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

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

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

//#ввод/вывод

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

for (int i = 0; i < obj.GetSize(); i++) {

istr >> obj.vector[i];

}

return istr;

}

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

cout << "(";

for (int i = 0; i < obj.size; i++) {

ostr << obj.vector[i];

if (i == obj.size - 1) { continue; }

cout << ",";

}

cout << ")" << endl;

return ostr;

}

};

Поля:

Size- размер вектора

start\_index – индекс, с которого выделяется память под вектор

vector – память для хранения элементов вектора

Методы:

TVector(int size\_ = 10, int start\_index\_ = 0);

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

Входные параметры: size – размер вектора, start\_index – стартовый индекс

Выходные параметры: отсутствуют

TVector(const TVector<Type>& obj);

Назначение: копирование векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: отсутствуют

~TVector();

Назначение: освобождение памяти vector

Входные параметры: отсутствуют

Выходные параметры: отсутствуют

int GetSize() const;

Назначение: получение размера вектора

Входные параметры: отсутствуют

Выходные параметры: size – размер вектора

int GetStart() const;

Назначение: получение стартового индекса

Входные параметры: отсутствуют

Выходные параметры: start\_index – стартовый индекс

Type& operator[](const int index);

Назначение: получение элемента памяти vector

Входные параметры: index – номер элемента

Выходные параметры: элемент с номером index-start\_index

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

Назначение: сравнение на равенство векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: целое число – 0 или 1

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

Назначение: сравнение на равенство векторов

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: целое число – 0 или 1

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

Назначение: присваивание полей

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: ссылка на объект себя

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

Назначение: сложить вектор со скаляром

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат сложения

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

Назначение: вычесть из вектора скаляр

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат вычитания

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

Назначение: умножить вектор на скаляр

Входные параметры: константная ссылка на скаляр

Выходные параметры: новый объект класса TVector как результат умножения

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

Назначение: сложить векторы

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат сложения

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

Назначение: вычесть один вектор из другого

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат вычитания

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

Назначение: умножить один вектор на другой

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: новый объект класса TVector как результат умножения

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

Назначение: ввод элементов вектора

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

Выходные параметры: istr- ссылка на стандартный поток ввода

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

Назначение: вывод элементов вектора

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

Выходные параметры: ostr- ссылка на стандартный поток вывода

### Описание класса TMatrix

template <class Type> class TMatrix : public TVector<TVector<Type>> {

public:

//#конструкторы

TMatrix(int mn = 10);

TMatrix(const TMatrix<Type>& matr);

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

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

//#сравнение матриц

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

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

//матричное-матричные операции

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

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

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

//#ввод/вывод

friend istream& operator>>(istream& istr, TMatrix& obj) {

for (int i = 0; i < obj.GetSize(); i++) {

for (int k = obj.GetStart() + i; k < obj.GetSize(); k++) {

cin >> obj[i][k];

}

}

return istr;

}

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

for (int i = 0; i < obj.GetSize(); i++) {

for (int j = 0; j < obj.GetStart() + i; j++) {

ostr << " ";

}

for (int k = obj.GetStart()+i; k < obj.GetSize(); k++) {

ostr << obj.vector[i][k] << " ";

}

ostr << endl;

}

return ostr;

}

};

Методы:

TMatrix(int mn = 10);

Назначение: выделение памяти под каждый вектор вектора векторов

Входные параметры: mn – размерность матрицы

Выходные параметры: отсутствуют

TMatrix(const TMatrix<Type>& matr);

Назначение: копирование матриц

Входные параметры: константная ссылка на объект класса TVector

Выходные параметры: отсутствуют

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

Назначение: преобразует вектор векторов в матрицу

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

Выходные параметры: отсутствуют

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

Назначение: присваивание матриц

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: ссылка объект себя

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

Назначение: проверка матриц на равенство

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: целое число – 0 или 1

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

Назначение: проверка матриц на неравенство

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: целое число – 0 или 1

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

Назначение: сложить матрицы

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: новый объект класса TMatrix как результат сложения

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

Назначение: вычесть из одной матрицы другую

Входные параметры: константная ссылка на объект класса TMatrix

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

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

Назначение: умножить матрицу на матрицу

Входные параметры: константная ссылка на объект класса TMatrix

Выходные параметры: константная ссылка на объект класса TMatrixкак результат умножения

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

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

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

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

# Заключение

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

.

# Литература

1. ([с.6](https://elar.urfu.ru/bitstream/10995/78551/1/978-5-7996-2776-8_2019.pdf))Н.В Гредасов, Линейная Алгебра – Издательство о Уральского университета.Редакционно-издательский отдел ИПЦ УрФУ 620049, 92 с.
2. ([с.6](https://www.chemometrics.ru/old/Tutorials/matrix.htm#ch106))Алексей Померанцев, Векторы и матрицы – Российское Хемометрическое общество,33 раздела

# 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;

}