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 механик**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«ВЕРХНЕТРЕУГОЛЬНЫЕ МАТРИЦЫ»**

**Выполнил(а):** студент группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шкребко М.С. /

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc152095736)

[1 Постановка задачи 4](#_Toc152095737)

[2 Руководство пользователя 5](#_Toc152095738)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc152095739)

[2.2 Приложение для демонстрации работы множеств 6](#_Toc152095740)

[3 Руководство программиста 7](#_Toc152095741)

[3.1 Алгоритмы 7](#_Toc152095742)

[3.1.1 Вектор 7](#_Toc152095743)

[3.1.2 Матрица 8](#_Toc152095744)

[3.2 Описание классов 8](#_Toc152095745)

[3.2.1 Класс TVector 8](#_Toc152095746)

[3.2.2 Класс TMatrix 11](#_Toc152095747)

[Заключение 14](#_Toc152095748)

[Литературы 15](#_Toc152095749)

[Приложения 16](#_Toc152095750)

[Приложение А. Реализация класса TVector 16](#_Toc152095751)

[Приложение Б. Реализация класса TMatrix 18](#_Toc152095752)

[Приложение В. Использование функционала векторов 19](#_Toc152095753)

[}Приложение Г. Использование функционала матриц 20](#_Toc152095754)

# Введение

Вектор – это математический объект, представляющие собой упорядоченный набор чисел, которые называются компонентами или координатами вектора. Для векторов определены операции сложения и разности двух векторов, умножения вектора на число. В евклидовом пространстве определена операция скалярного произведения двух векторов.

Одно из применений векторов – это представление матриц. Матрица – это прямоугольная таблица определённого размера, содержащая действительные числа в качестве элементов. Один из способов представления матриц является упорядоченный набор векторов.

Одним из видов матриц, является верхнетреугольная матрица. Это квадратная матрица, в которой все элементы ниже главной диагонали равны 0. Лучше использовать такую матрицу, так как хранить в памяти необходимо только половину матрицы, и это сократит вдвое затраты по памяти, в сравнении с полной матрицей.

# Постановка задачи

Цель – получить практические навыки применения структуры данных вектора и матрицы, получить практические знания использования шаблона.

Задачи:

1. Разработать классы, представляющие вектор и матрицу.
2. Реализовать все необходимые операции с векторами и матрицами.
3. Протестировать корректность работы разработанного функционала.
4. Сделать вывод о проделанной работе.

# Руководство пользователя

## Приложение для демонстрации работы векторов

1. Запустить sample\_tvector.exe. В результате появится следующее окно. ():

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

1. Стартовое окно приложения
2. Приложение требует создать третий вектор. Ещё три вектора были созданы заранее программой. Нужно ввести размер и координаты. ().

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

1. Окно создания вектора
2. Выводится результат векторных операций и заканчивается программа. ().

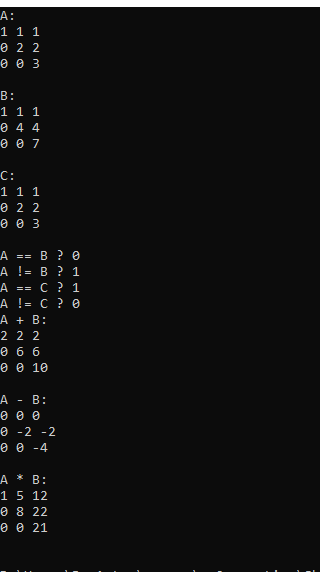
Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

1. Результаты векторных операций

## Приложение для демонстрации работы множеств

1. Запустить sample\_tmatrix.exe. В результате появится следующее окно. Все матрицы сразу создаются размером 3 на 3 и заполняются фиксированными числами. ().



1. Стартовое окно приложения

# Руководство программиста

## Алгоритмы

### Вектор

Вектор – структура хранения. Он хранит элементы одного типа данных.

Вектор хранится в виде массива элементов, стартового индекса и количества элементов в векторе. Такая структура позволяет эффективно работать с матричными операциями.

Если стартовый индекс отличен от нуля, то все элементы от 0 до стартового индекса (не включительно) будут равны нейтральному элементу (нулю).

Вектор поддерживает операции сложения, вычитания и умножения с элементом типа данных, сложения, вычитания, скалярного произведения с вектором того же типа данных, операции индексации, сравнение на равенство (неравенство).

**Операция сложения**

Операция сложения определена для вектора того же типа (складываются элементы первого и второго вектора с одинаковыми индексами) или некоторого элемента того же типа (каждый элемент вектора отдельно складывается с элементом).

**Операция вычитания**

Операция вычитания определена для вектора того же типа (вычитаются элементы первого и второго вектора с одинаковыми индексами) или некоторого элемента того же типа (каждый элемент вектора отдельно вычитается с элементом).

**Операция умножения**

Операция умножения определена для вектора того же типа (скалярное произведение векторов) или некоторого элемента того же типа (каждый элемент вектора отдельно умножается с элементом).

**Операция индексации**

Операция индексации предназначена для получения элемента вектора. Причем, если позиция будет меньше, чем стартовый индекс, то будет выведено исключение.

**Операция сравнения на равенство**

Операция сравнения на равенство с вектором возвращает 1, если вектора равны поэлементно, причём их стартовые индексы и размеры тоже равны, 0 в противном случае.

**Операция сравнения на неравенство**

Операция сравнения на неравенство с вектором возвращает 0, если вектора не равны поэлементно, или если их стартовые индексы не равны, 1 в противном случае.

### Матрица

Матрица – вектор векторов, структура хранения. Она хранит элементы одного типа данных.

Матрица хранится в виде массива векторов, стартового индекса и количества элементов в матрице .

Матрица поддерживает операции сложения, вычитания и умножения с матрицей того же типа данных, операции индексации, сравнение на равенство.

**Операция сложения**

Операция сложения определена для матрицы того же типа (складываются элементы первой и второй матрицы с одинаковыми индексами).

**Операция вычитания**

Операция вычитания определена для матрицы того же типа (вычитаются элементы первой и второй матрицы с одинаковыми индексами).

**Операция умножения**

Операция умножения определена для матрицы того же типа (скалярное произведение векторов).

**Операция индексации**

Операция индексации предназначена для получения элемента матрицы. Причем, Элемент матрицы – вектор-строка, также можно вывести элемент матрицы по индексу, т.к. для вектора также перегружена операция индексации.

**Операция сравнения на равенство**

Операция сравнения на равенство с матрицей возвращает 1, если они равны поэлементно, причём их стартовые индексы и размеры тоже равны, 0 в противном случае.

## Описание классов

### Класс TVector

Объявление класса:

template <typename ValueType> class TVector

{

protected:

int Size;

int StartIndex;

ValueType\* pVector;

public:

TVector(int s = 10, int i = 0);

TVector(const TVector<ValueType>& vec);

~TVector();

int GetSize() const;

int GetStartIndex() const;

ValueType& operator[](const int i);

int operator==(const TVector<ValueType>& v) const;

int operator!=(const TVector<ValueType>& v) const;

TVector operator\*(const ValueType& v);

TVector operator+(const ValueType& v);

TVector operator-(const ValueType& v);

TVector operator+(const TVector<ValueType>& v);

double operator\*(const TVector<ValueType>& v);

TVector operator-(const TVector<ValueType>& v);

const TVector& operator=(const TVector<ValueType>& v);

friend istream& operator>>(istream& istr, TVector<ValueType>& v)

friend ostream& operator<<(ostream& ostr, TVector<ValueType>& v)

};

Поля:

size – размер вектора.

StartIndex – стартовый индекс вектора.

pVector – указатель типа ValueType на первый элемент вектора.

Методы:

int GetSize() const;

Назначение: получение пользователем размера вектора.

Выходные данные: целое число – размер вектора.

int GetStartIndex() const;

Назначение: получение стартового индекса вектора.

Выходные данные: целое число – стартовый индекс вектора.

*ValueType*& TVector<*ValueType*>::operator[] (const int *i*);

Назначение: получить элемент вектора под конкретным номером.

Входные данные: номер элемента (индекс) в векторе (целое число от 0 до значения размера вектора не включительно).

Выходные данные: значение элемента вектора под переданным номером.

int operator== (const TVector<*ValueType*>& *v*) const;

Назначение: сравнение двух векторов.

Входные данные: вектор, с которым необходимо сравнить текущий.

Выходные данные: результат сравнения (1, если равны; 0, если не равны).

int operator!= (const TVector<*ValueType*>& *v*) const;

Назначение: сравнение двух векторов.

Входные данные: вектор, с которым необходимо сравнить текущий.

Выходные данные: результат сравнения (0, если равны; 1, если не равны).

const *TVector*& operator= (const TVector<*ValueType*>& *v*);

Назначение: присвоение (копирование) переданного вектора в текущий.

Входные данные: вектор, который будет скопирован.

Выходные данные: результат копирования – ссылка на копию копируемого вектора.

*TVector* operator+ (const *ValueType* *v*);

Назначение: сложение вектора со скаляром.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат сложения.

*TVector* operator- (const *ValueType* *v*);

Назначение: вычитание из вектора скаляра.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат вычитания.

*TVector* operator\* (const *ValueType* *v*);

Назначение: умножения вектора на скаляр.

Входные данные: скаляр (числовое значение).

Выходные данные: результирующий вектор – результат умножения.

*TVector* operator+ (const TVector<*ValueType*>& *v*);

Назначение: сложение двух векторов.

Входные данные: вектор.

Выходные данные: результирующий вектор – результат сложения исходного и переданного вектора.

*TVector* operator- (const TVector<*ValueType*>& *v*;

Назначение: вычитание двух векторов.

Входные данные: вектор.

Выходные данные: результирующий вектор – результат вычитания исходного вектора и переданного.

double operator\* (const TVector<*ValueType*>& *v*);

Назначение: вычисление скалярного произведения векторов.

Входные данные: вектор.

Выходные данные: вещественное число – результат скалярного произведения двух векторов.

**friend istream& operator>>(istream& istr, TVector<ValueType>& v)**

Назначение: ввод вектора.

Входные данные: ссылка на поток ввода и вектор.

Выходные данные: ссылка на поток ввода.

friend std::*ostream*& operator<< (std::*ostream*& *out*, *TVector*<*ValueType*>& *v*);

Назначение: вывод вектора.

Входные данные: ссылка на поток вывода и вектор.

Выходные данные: ссылка на поток вывода.

### Класс TMatrix

template <typename ValueType>

class TMatrix : public TVector<TVector<ValueType>>

{

public:

TMatrix(int n = 10);

TMatrix(const TMatrix& m);

TMatrix(const TVector <TVector<ValueType>>& m);

const TMatrix& operator=(const TMatrix& m);

int operator==(const TMatrix& m) const;

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

TMatrix operator+(const TMatrix& m);

TMatrix operator-(const TMatrix& m);

TMatrix operator\*(const TMatrix& m);

friend istream& operator>>(istream& istr, TMatrix<ValueType>& m)

friend ostream& operator<<(ostream& ostr, TMatrix<ValueType>& m)

Методы:

int operator== (const TMatrix<*ValueType*>& *m*) const;

Назначение: сравнение двух матриц.

Входные данные: матрица, с которой будет производится сравнение.

Выходные данные: результат сравнения (1, если матрицы равны; 0, если матрицы не равны).

int operator!= (const TMatrix<*ValueType*>& *m*) const;

Назначение: сравнение двух матриц.

Входные данные: матрица, с которой будет производится сравнение.

Выходные данные: результат сравнения (0, если матрицы равны; 1, если матрицы не равны).

const *TMatrix*& operator= (const TMatrix<*ValueType*>& *m*);

Назначение: копирование переданной матрицы в исходную.

Входные данные: копируемая матрица.

Выходные данные: матрица – копия переданной матрицы.

*TMatrix* operator+ (const TMatrix<*ValueType*>& *m*);

Назначение: Сложение двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат сложения исходной матрицы и переданной.

*TMatrix* operator- (const TMatrix<*ValueType*>& *m*);

Назначение: вычитание двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат вычитания исходной матрицы и переданной.

*TMatrix* operator\* (const TMatrix<*ValueType*>& *m*);

Назначение: умножение двух матриц.

Входные данные: матрица.

Выходные данные: результирующая матрицы – результат умножения исходной матрицы и переданной.

friend istream& operator>>(istream& istr, TMatrix<ValueType>& m);

Назначение: ввод данных матрицы (размер и элементов).

Входные данные: поток ввода и матрица.

Выходные данные: поток ввода.

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

Назначение: вывод матрицы на экран.

Входные данные: поток вывода и матрица.

Выходные данные: поток вывода.

# Заключение

В результате данной лабораторной работы были получены практические навыки применения векторов в верхнетреугольных матрицах. На основе полученных знаний в этой предметной области была разработана программа, которая реализует хранение векторов и верхнетреугольных матриц, а также операции над ними. Была произведена проверка корректной работы реализованного функционала с различными наборами данных. В целом, лабораторная работа помогла понять основные принципы работы с верхнетреугольными матрицами и векторами.

В ходе выполнения лабораторной работы мы изучили и практически применили концепцию шаблонов в языке программирования C++. Шаблоны позволяют создавать обобщенные типы данных, которые могут быть использованы с различными типами данных без необходимости дублирования кода.

В рамках работы мы разработали шаблонный класс для реализации вектора, который поддерживает основные операции, такие как добавление элемента, удаление элемента, доступ к элементу по индексу и другие. Также мы разработали шаблонный класс для реализации верхнетреугольной матрицы, который поддерживает операции сложения матриц, умножения матрицы на матрицу и другие

# Литературы

# Приложения

## Приложение А. Реализация класса TVector

template <typename ValueType>

TVector<ValueType> ::TVector(int s, int i)

{

if (s < 0)

throw "out of range";

else

if (s == 0)

{

Size = s;

pVector = nullptr;

}

StartIndex = i;

Size = s;

pVector = new ValueType[Size];

for (int i = 0; i < Size; i++)

pVector[i] = 0;

}

template <typename ValueType>

TVector<ValueType> ::TVector(const TVector<ValueType>& v)

{

Size = v.Size;

StartIndex = v.StartIndex;

pVector = new ValueType[Size];

for (int i = 0; i < Size; i++)

{

pVector[i] = v.pVector[i];

}

}

template <typename ValueType>

TVector<ValueType>::~TVector()

{

delete[] pVector;

}

template <typename ValueType>

int TVector<ValueType>::GetSize() const

{

return Size;

}

template <typename ValueType>

int TVector<ValueType>::GetStartIndex() const

{

return StartIndex;

}

template <typename ValueType>

ValueType& TVector<ValueType> :: operator [](const int i)

{

if (i < 0 || i >= Size)

throw "out of range";

return pVector[i];

}

template <typename ValueType>

int TVector<ValueType>::operator==(const TVector<ValueType>& v) const

{

if (Size != v.Size || StartIndex != v.StartIndex) {

return 0;

}

for (int i = 0; i < Size; i++) {

if (pVector[i] != v.pVector[i]) {

return 0;

}

}

return 1;

}

template <typename ValueType>

int TVector<ValueType>::operator!=(const TVector<ValueType>& v) const

{

return !((\*this) == v);

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator\*(const ValueType& v)

{

TVector<ValueType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp[i] = pVector[i] \* v;

return tmp;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator+(const ValueType& n)

{

TVector<ValueType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp[i] = pVector[i] + n;

return tmp;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>::operator-(const ValueType& n)

{

TVector<ValueType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

tmp[i] = pVector[i] - n;

return tmp;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>:: operator+(const TVector<ValueType>& v)

{

if ((Size != v.Size) || (StartIndex != v.StartIndex))

throw "diff size or start index";

TVector<ValueType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

{

tmp.pVector[i] = pVector[i] + v.pVector[i];

}

return tmp;

}

template <typename ValueType>

TVector<ValueType> TVector<ValueType>:: operator-(const TVector<ValueType>& v)

{

if ((Size != v.Size) || (StartIndex != v.StartIndex))

throw "diff size";

TVector<ValueType> tmp(Size, StartIndex);

for (int i = 0; i < Size; i++)

{

tmp.pVector[i] = pVector[i] - v.pVector[i];

}

return tmp;

}

template <typename ValueType>

double TVector<ValueType>::operator\*(const TVector<ValueType>& v)

{

if ((Size != v.Size) || (StartIndex != v.StartIndex))

throw "diff size";

double sum = 0.0;

for (int i = 0; i < Size; i++)

{

sum += pVector[i] \* v.pVector[i];

}

return sum;

}

template <typename ValueType>

const TVector<ValueType>& TVector<ValueType>::operator=(const TVector<ValueType>& v)

{

if (this == &v)

return \*this;

if (Size != v.Size)

{

delete[] pVector;

Size = v.Size;

pVector = new ValueType[Size];

}

StartIndex = v.StartIndex;

for (int i = 0; i < Size; i++)

{

pVector[i] = v.pVector[i];

}

return \*this;

}

## Приложение Б. Реализация класса TMatrix

template <typename ValueType>

TMatrix<ValueType>::TMatrix(int n) : TVector<TVector<ValueType>>(n)

{

for (int i = 0; i < n; i++)

{

TVector<TVector<ValueType>>::pVector[i] = TVector<ValueType>(n - i, i);

}

}

template<typename ValueType>

TMatrix<ValueType>::TMatrix(const TMatrix& m) : TVector<TVector<ValueType>>(m) { };

template <typename ValueType>

TMatrix<ValueType>::TMatrix(const TVector<TVector<ValueType>>& m) :TVector<TVector<ValueType>>(m) { };

template<typename ValueType>

const TMatrix<ValueType>& TMatrix<ValueType>::operator=(const TMatrix<ValueType>& m)

{

return TVector<TVector<ValueType>>::operator=(m);

}

template <typename ValueType>

int TMatrix<ValueType>::operator==(const TMatrix& m) const

{

return TVector<TVector<ValueType> >::operator==(m);

}

template <typename ValueType>

int TMatrix<ValueType>::operator!=(const TMatrix& m) const

{

return TVector<TVector<ValueType> >::operator!=(m);

}

template<typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator+(const TMatrix& m)

{

return TVector<TVector<ValueType>>::operator+(m);

}

template<typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator-(const TMatrix& m)

{

return TVector<TVector<ValueType> > :: operator-(m);

}

template<typename ValueType>

TMatrix<ValueType> TMatrix<ValueType>::operator\*(const TMatrix& m)

{

if (this->Size != m.Size)

throw "diff size\n";

TMatrix<ValueType> tmp(this->Size);

for (int i = 0; i < this->Size; i++)

{

for (int j = i; j < this->Size; j++)

{

for (int k = i; k <= j; k++)

{

tmp[i][j - i] += this->pVector[i][k - i] \* m.pVector[k][j - k];

}

}

}

return tmp;

}

## Приложение В. Использование функционала векторов

#include <iostream>

#include "TVector.h"

int main()

{

TVector<int> v1(4, 1);

TVector<int> v2(4, 1);

TVector<int> v3;

TVector<int> v4(4, 1);

for (int i = 0; i < 4; i++)

{

v1[i] = i + 1;

v2[i] = i \* 3 + 1;

v4[i] = v2[i];

}

cin >> v3;

std::cout << "v1 = " << v1 << '\n';

std::cout << "v2 = " << v2 << '\n';

std::cout << "v3 = " << v3 << '\n';

std::cout << "v4 = " << v4 << '\n' << '\n';

std::cout << "Size of v1 = v2 = " << v1.GetSize() << '\n';

std::cout << "Size of v3 = " << v3.GetSize() << '\n' << '\n';

std::cout << "v1 == v2? " << (v1 == v2) << '\n';

std::cout << "v2 != v4? " << (v2 != v4) << '\n';

std::cout << "v2 == v4? " << (v2 == v4) << '\n' << '\n';

std::cout << "v1 \* 5 = " << v1 \* 5 << '\n';

std::cout << "v1 + 5 = " << v1 + 5 << '\n';

std::cout << "v1 - 5 = " << v1 - 5 << '\n' << '\n';

std::cout << "v1 + v2 = " << v1 + v2 << '\n';

std::cout << "v2 - v1 = " << v2 - v1 << '\n';

std::cout << "v1 scalar \* v2 = " << v1 \* v2 << '\n';

return 0;

## }Приложение Г. Использование функционала матриц

#include <iostream>

#include "TMatrix.h"

int main()

{

TMatrix <double> A(3), B(3), C(3);

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3 - i; j++)

{

A[i][j] = 1 + i;

B[i][j] = 1 + 3 \* i;

C[i][j] = A[i][j];

}

std::cout << "A:" << std::endl << A << std::endl;

std::cout << "B:" << std::endl << B << std::endl;

std::cout << "C:" << std::endl << C << std::endl;

std::cout << "A == B ? " << (A == B) << std::endl;

std::cout << "A != B ? " << (A != B) << std::endl;

std::cout << "A == C ? " << (A == C) << std::endl;

std::cout << "A != C ? " << (A != C) << std::endl;

std::cout << "A + B:" << std::endl << A + B << std::endl;

std::cout << "A - B:" << std::endl << A - B << std::endl;

std::cout << "A \* B:" << std::endl << A \* B << std::endl;

return 0;

}