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:

«Вектор и верхнетреугольная матрица»

Выполнил(а): студент(ка) группы 3822Б1ФИ2
_____ / Рысев М.Д./
Подпись

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

Нижний Новгород
2023

Содержание

Введение

В программировании часто приходиться работать с такими алгебраическими объектами как вектора и матрицы. Матрицы могут быть разного вида: квадратные, прямоугольные, единичные, разряжённые и т.д. Из всего разнообразия мы рассмотрим верхне-треугольные матрицы - матрицы, в которых сверху от главной диагонали находятся любые допустимые числа, а снизу только нули. Хранить такую матрицу мы будем в виде вектора векторов.

1 Постановка задачи

1. Вспомнить основные понятия касательно векторов и матриц.
2. Реализовать класс для представления векторов.
3. Реализовать класс для представления верхне-треугольных матриц.
4. Написать тесты для проверки работоспособности классов.
5. Написать программы для демонстрации работы написанных классов.

2 Руководство пользователя

2.1 Приложение для демонстрации работы векторов

Запустите программу sample_vec.exe. Запуститься консоль, в которой необходимо, следуя указаниям, ввести два вектора (рис. 1).

```
D:\mp2-practice\RysevMD\02_lab\sln\bin>sample_vec.exe
Enter vec1 (size = 5): 1 2 3 4 5
Enter vec2 (size = 5): 5 4 3 2 1
```

Рис. 1. Ввод векторов.

При нажатии на клавишу Enter программа выведет результаты векторно-векторных, векторно-скалярных и логических операций (рис. 2).

```
vector-vector operations
vec1 + vec2 =  6  6  6  6  6
vec1 - vec2 =  -4  -2  0  2  4
vec1 * vec2 = 35

vector-scalar operations
vec1 + 15 =  16  17  18  19  20
vec2 - 10 =  -5  -6  -7  -8  -9
vec1 * 5 =  5  10  15  20  25

bool operations
vec1 == vec2? - 0
vec1 != vec2? - 1
```

Рис. 2. Результаты операций.

2.2 Приложение для демонстрации работы матриц

Запустите программу sample_mtrx.exe. Запуститься консоль, в которой необходимо, следуя указаниям, ввести две матрицы (рис. 3).

```
D:\mp2-practice\RysevMD\02_lab\sln\bin>sample_mtrx.exe
matrix input format (size = n):
a11 a12 a13 ... a1n
    a22 a23 ... a2n
        a33 ... a3n
...................
                ann

Enter mtr1 (size = 3):
1 2 3
  4 5
    6
Enter mtr2 (size = 3):
6 5 4
  3 2
    1
```

Рис. 3. Ввод матриц.

При нажатии на клавишу Enter программа выведет результаты матрично-матричных и логических операций (рис. 4).

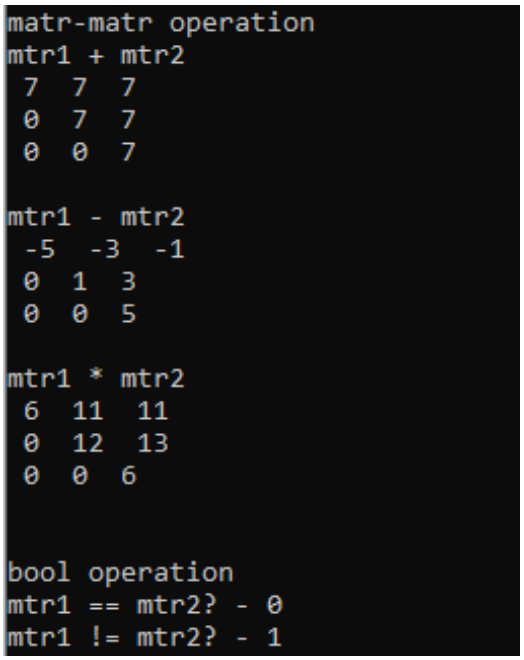

Рис. 4. Результаты операций.

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Вектора

В нашем случае вектор храниться как массив элементов одного типа. Пример целочисленного вектора размера 5:

$$V = (1\ 2\ 3\ 4\ 5)$$

В рамках работы реализованы следующие операции: сложение векторов, разность векторов, произведение векторов, сложение вектора с скаляром, разность вектора со скаляром, умножение вектора на скаляр, сравнение векторов на равенство и на неравенство.

Сложение векторов.

Создаётся результирующий вектор, в каждую компоненту которого записывается результат сложения соответствующих компонент исходных векторов. Вектора должны быть одного размера. В противном случае программа бросает исключение.

Пример:

$$A = (1\ 2\ 3)$$

$$B = (4\ 5\ 6)$$

$$A + B = (5\ 7\ 9)$$

Разность векторов.

Создаётся результирующий вектор, в каждую компоненту которого записывается результат разности соответствующих компонент исходных векторов. Вектора должны быть одного размера. В противном случае программа бросает исключение.

Пример:

$$A = (12\ -9\ 7)$$

$$B = (15\ 31\ 6)$$

$$B - A = (3,\ 40,\ -1)$$

Произведение векторов.

Суммируются произведения соответствующих компонент исходных векторов. Вектора должны быть одного размера. В противном случае программа бросает исключение.

Пример:

$$A = (1\ 2\ 3)$$

$$B = (3\ 2\ 1)$$

$$A \cdot B = (1 \cdot 3) + (2 \cdot 2) + (3\ \ 1) = 3 + 4 + 3 = 10$$

Сумма вектора и скаляра.

К каждой компоненте вектора прибавляется скаляр.

Пример:

$$A = (1\ 2\ 3)$$

$$A + 10 = (1 + 10\ \ \ 2 + 10\ \ \ 3 + 10) = (11\ 12\ 13)$$

Разность вектора и скаляра.

Из каждой компоненты вектора вычитается скаляр.

Пример:

$$A = (10\ 20\ 30)$$

$$A - 15 = (10 - 15\ \ \ 20 - 15\ \ \ 30 - 15) = (-5, 5, 15)$$

Умножение вектора на скаляр.

Каждая компонента вектора умножается на скаляр.

Пример:

$$A = (7, 8, 9)$$

$$A \cdot 10 = (7 \cdot 10\ \ \ 8 \cdot 10\ \ \ 9 \cdot 10) = (70\ 80\ 90)$$

Сравнение векторов на равенство.

Если все соответствующие компоненты двух векторов равны, то и векторы равны. В противном случае вектора не равны.

Пример:

$$A = (1\ 2\ 3)$$

$$B = (1\ 2\ 3)$$

$$C = (1\ 2\ 2)$$

$$A == B \text{ - Да}$$

$$A == C \text{ - Нет}$$

Сравнение векторов на неравенство.

Если все соответствующие компоненты двух векторов равны, то и векторы равны. В противном случае вектора не равны.

Пример:

A = (1 2 3)

B = (1 2 3)

C = (1 2 2)

A ≠ B - Нет

A ≠ C - Да

3.1.2 Матрицы

В нашем случае матрица храниться как вектор векторов.

Пример трёхмерной верхнетреугольной матрицы :

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

Каждая строка является вектором. Каждый вектор является компонентой того вектора, в котором храниться вся матрица. Абстрактный вид:

M = ((a_{11} a_{12} ... a_{1n})
(a_{22} ... a_{2n})
(a_{nn}))

Для класса матриц реализованы следующие операции: сумма матриц, разность матриц, произведение матриц, сравнение матриц на равенство и на неравенство. В нашем случае класс матриц наследуется от класса векторов. Это означает, что все перечисленные выше операции, за исключением умножения, уже реализованы в базовом классе.

Сумма матриц.

Создаётся результирующая матрица, в каждую строку которой записывается сумма соответствующих строк исходных матриц.

Пример:

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

$$A + B = \begin{pmatrix} 4 & 4 \\ 0 & 4 \end{pmatrix}$$

Разность матриц.

Создаётся результирующая матрица, в каждую строку которой записывается разность соответствующих строк исходных матриц.

Пример:

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

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

Произведение матриц.

Создаётся результирующая матрица, каждый элемент которой вычисляется по формуле (т.к матрица верхнетреугольная, то данная формула отличается от формулы для матриц произвольного вида):

$$c_{ij} = \sum_{k=i}^{j} a_{ik} b_{kj}$$

где a,b – элементы матриц множителей, i – номер строки, j – номер столбца.

Пример:

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

$$A \cdot B = \begin{pmatrix} 3 & 4 \\ 0 & 3 \end{pmatrix}$$

Сравнение матриц на равенство

Операция сравнения матриц на равенство возвращает 1, если матрицы покомпонентно равны, и 0 в противном случае.

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

$$A == B - 0$$

Сравнение матриц на неравенство

Операция сравнения матриц на неравенство возвращает 1, если матрицы покомпонентно не равны, и 0 в противном случае.

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

$$A \neq B - 1$$

3.2 Описание программной реализации

3.2.1 Описание класса TVector

```
template <typename T>
class TVector {
protected:
int size;
int StartIndex;
T* elems;
public:
TVector(int_size = 10, int_st = 0);
TVector(const TVector<T>& vec);
virtual ~TVector();
int GetSize() const;
int GetStartIndex() const;
T& operator[] (const int ind);
friend std::istream& operator >> (std::istream& in, TVector& vec) {
for (int i = 0; i < vec.GetSize(); i++) in >> vec.elems[i];
return in;
}
friend ostream& operator << (ostream& out, TVector& vec) {
for (int i = 0; i < vec.GetStartIndex(); i++) out << " 0 ";
for (int i = 0; i < vec.GetSize(); i++) out << " " << vec.elems[i] << "
";
return out;
}
const TVector<T>& operator = (const TVector<T>& vec);
TVector<T> operator + (const TVector<T>& vec) const;
TVector<T> operator - (const TVector<T>& vec) const;
T operator * (const TVector<T>& vec) const;
TVector<T> operator + (const T num) const;
TVector<T> operator - (const T num) const;
TVector<T> operator * (const T num) const;
int operator == (const TVector<T>& vec) const;
int operator != (const TVector<T>& vec) const;
};
```

Назначение – представление вектора.

Поля:

size - размер вектора.

StartIndex – стартовый индекс – указывает на какой строке в матрице находиться вектор.

elems – массив с элементами вектора.

Методы:

TVector(int_size = 10, int_st = 0);
Назначение - конструктор класса.
Входные параметры:
int_size – количество элементов вектора.
int_st – стартовый индекс.

TVector(const TVector<T>& vec);
Назначение – конструктор копирования.
Входные параметры:

vec – ссылка на вектор.

```
virtual ~TVector();
```

Назначение – деструктор – очищает динамически выделенную память.

```
int GetSize() const;
```

Назначение – возвращение размера вектора.

Выходные параметры – количество элементов вектора.

```
int GetStartIndex() const;
```

Назначение – возвращение стартового индекса.

Выходные параметры – стартовый индекс.

```
T& operator[] (const int ind);
```

Назначение – операция индексации.

Входные параметры:

ind – индекс элемента.

Выходные параметры – ссылка на индексируемый элемент.

```
friend std::istream& operator >> (std::istream& in, TVector& vec);
```

Назначение – ввод вектора .

Входные параметры:

in – ссылка на поток ввода.

vec – ссылка на вектор.

Выходные параметры – ссылка на поток ввода.

```
friend ostream& operator << (ostream& out, TVector& vec);
```

Назначение – вывод вектора.

Входные параметры:

out – ссылка на поток вывода.

vec – ссылка на вектор.

Выходные параметры – ссылка на поток вывода.

```
const TVector<T>& operator = (const TVector<T>& vec);
```

Назначение – оператор присваивания.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – ссылка на вектор.

```
TVector<T> operator + (const TVector<T>& vec) const;
```

Назначение – сложение векторов.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – ссылка на вектор.

```
TVector<T> operator - (const TVector<T>& vec) const;
```

Назначение – вычитание векторов.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – ссылка на вектор.

```
T operator * (const TVector<T>& vec) const;
```
Назначение – произведение векторов.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – число – результат произведения.

```
TVector<T> operator + (const T num) const;
```
Назначение – прибавление числа к вектору.

Входные параметры:

num – число.

Выходные параметры – вектор – результат сложения.

```
TVector<T> operator - (const T num) const;
```
Назначение – вычитание числа из вектора.

Входные параметры:

num – число.

Выходные параметры – вектор – результат вычитания.

```
TVector<T> operator * (const T num) const;
```
Назначение – умножение вектора на число.

Входные параметры:

num – число.

Выходные параметры – вектор – результат умножения.

```
int operator == (const TVector<T>& vec) const;
```
Назначение – сравнение векторов на равенство.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – 1, если векторы равны, иначе 0.

```
int operator != (const TVector<T>& vec) const;
```
Назначение – сравнение векторов на неравенство.

Входные параметры:

vec – ссылка на вектор.

Выходные параметры – 1, если векторы неравны, иначе 0.

3.2.2 Описание класса TMatrix

```
template <typename T>
class TMatrix : public TVector<TVector<T>> {
public:
TMatrix(int N = 5);
TMatrix(const TMatrix<T>& matr);
TMatrix(const TVector<TVector<T>>& vecs);
const TMatrix<T>& operator = (const TMatrix<T>& matr);
int operator == (const TMatrix<T>& matr) const;
int operator != (const TMatrix<T>& matr) const;
TMatrix<T> operator + (const TMatrix<T>& matr);
TMatrix<T> operator - (const TMatrix<T>& matr);
TMatrix<T> operator * (const TMatrix<T>& matr);
friend istream& operator >> (istream& in, TMatrix<T>& matr) {
for (int i = 0; i < matr.GetSize(); i++) in >> matr.elems[i];
return in;
}
friend ostream& operator << (ostream& out, const TMatrix<T>& matr) {
for (int i = 0; i < matr.GetSize(); i++) out << matr.elems[i] << "\n";
return out;
}
};
```

Назначение – представление матриц.

Поля:

`TMatrix(int N = 5);`

Назначение – конструктор.

Входные параметры:

`N` – размерность матрицы.

`TMatrix(const TMatrix<T>& matr);`

Назначение – конструктор копирования.

Входные параметры:

`matr` – ссылка на матрицу.

`TMatrix(const TVector<TVector<T>>& vecs);`

Назначение – конструктор преобразования типа от вектора векторов к матрице

Входные данные:

`&vecs` – ссылка на вектор векторов.

`const TMatrix<T>& operator = (const TMatrix<T>& matr);`

Назначение – оператор присваивания.

Входные параметры:

`matr` – ссылка на матрицу.

Выходные параметры – ссылка на матрицу.

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

Назначение – сравнение на равенство.

Входные параметры:

`matr` – ссылка на матрицу.

Выходные параметры – 1, если матрицы равны, иначе 0.

```
int operator != (const TMatrix<T>& matr) const;
```
Назначение – сравнение на неравенство.
Входные параметры:
matr – ссылка на матрицу.
Выходные параметры – 1, если матрицы неравны, иначе 0.

```
TMatrix<T> operator + (const TMatrix<T>& matr);
```
Назначение – сумма матриц.
Входные параметры:
matr – ссылка на матрицу.
Выходные параметры:
Результирующая матрица.

```
TMatrix<T> operator - (const TMatrix<T>& matr);
```
Назначение – разность матриц.
Входные параметры:
matr – ссылка на матрицу.
Выходные параметры:
Результирующая матрица.

```
TMatrix<T> operator * (const TMatrix<T>& matr);
```
Назначение – произведение матриц.
Входные параметры:
&matr – ссылка на матрицу.
Выходные параметры:
Результирующая матрица.

```
friend istream& operator >> (istream& in, TMatrix<T>& matr);
```
Назначение – ввод матриц.
Входные параметры:
in – ссылка на поток ввода.
matr – ссылка на матрицу.
Выходные параметры:
Ссылка на поток ввода.

```
friend ostream& operator << (ostream& out, const TMatrix<T>& matr);
```
Назначение – вывод матриц.
Входные параметры:
out – ссылка на поток вывода.
matr – ссылка на матрицу.
Выходные параметры:
Ссылка на поток вывода.

Заключение

В ходе работы выполнены следующие действия:

1. Освежены знания касательно векторов и матриц.
2. Реализован класс для представления векторов и матриц.
3. Реализован класс для представления верхнетреугольных матриц.
4. Написаны тесты для проверки работоспособности классов.
5. Написаны программы для демонстрации работы написанных классов.

Литература

1. mathprofi [http://mathprofi.ru/deistviya_s_matricami.html]

Приложения

Приложение А. Реализация класса TVector

```
template <typename T>
TVector<T>::TVector(int _size, int _st) {
	if (_size < 0 || _st < 0) throw "out_of_range";
	size = _size;
	StartIndex = _st;
	elems = new T[size];
}
template <typename T>
TVector<T>::TVector(const TVector<T>& vec) {
	size = vec.size;
	StartIndex = vec.StartIndex;
	elems = new T[size];
	for (int i = 0; i < size; i++) elems[i] = vec.elems[i];
}
template <typename T>
TVector<T>::~TVector() {
	if (size > 0) delete[] elems;
}
template <typename T>
int TVector<T>::GetSize() const {
	return size;
}
template <typename T>
int TVector<T>::GetStartIndex() const {
	return StartIndex;
}
template <typename T>
T& TVector<T>::operator[] (const int ind) {
	if (ind < 0 || ind > size + StartIndex) throw "out_of_range";
	if (ind < StartIndex) throw "elemen_not_exist";
	return elems[ind - StartIndex];
}
template <typename T>
const TVector<T>& TVector<T>::operator=(const TVector<T>& vec) {
	if ((*this) == vec) return (*this);
	if (size != vec.size) {
		size = vec.size;
		delete[] elems;
		elems = new T[size];
	}
	StartIndex = vec.StartIndex;
	for (int i = 0; i < size; i++) elems[i] = vec.elems[i];
	return (*this);
}
template<typename T>
TVector<T> TVector<T>::operator + (const TVector<T>& vec) const {
	if (size != vec.size) throw "Different sizes";
	if (StartIndex != vec.StartIndex) throw "Different start
indexes";

	TVector<T> res(vec.size, vec.StartIndex);
	for (int i = 0; i < size; i++)
		res.elems[i] = elems[i] + vec.elems[i];
	return res;
}
template<typename T>
```

```
TVector<T> TVector<T>::operator - (const TVector<T>& vec) const {
	if (size != vec.size) throw "Different sizes";
	if (StartIndex != vec.StartIndex) throw "Different start
indexes";

	TVector<T> res(vec.size, vec.StartIndex);
	for (int i = 0; i < size; i++)
		res.elems[i] = elems[i] - vec.elems[i];
	return res;
}
template <typename T>
T TVector<T>::operator * (const TVector<T>& vec) const {
	if (size != vec.size) throw "Different spaces";
	if (StartIndex != vec.StartIndex) throw "Different start
indexes";

	T res = 0;
	for (int i = 0; i < size; i++) res += elems[i] * vec.elems[i];
	return res;
}
template <typename T>
TVector<T> TVector<T>::operator + (const T num) const {
	TVector<T> res(*this);
	for (int i = 0; i < size; i++) res.elems[i] += num;
	return res;
}
template <typename T>
TVector<T> TVector<T>::operator - (const T num) const {
	TVector<T> res(*this);
	for (int i = 0; i < size; i++) res.elems[i] -= num;
	return res;
}
template <typename T>
TVector<T> TVector<T>::operator * (const T num) const {
	TVector<T> res(*this);
	for (int i = 0; i < size; i++) res.elems[i] *= num;
	return res;
}
template <typename T>
int TVector<T>::operator == (const TVector<T>& vec) const {
	if (size != vec.size || StartIndex != vec.StartIndex) return
false;
	for (int i = 0; i < size; i++) {
		if (elems[i] != vec.elems[i]) return false;
	}
	return true;
}
template <typename T>
int TVector<T>::operator != (const TVector<T>& vec) const {
	return !((*this) == vec);
}
```

Приложение Б. Реализация класса TMatrix

```
template <typename T>
TMatrix<T>::TMatrix(int N) : TVector<TVector<T>>(N) {
      for (int i = 0; i < size; i++) elems[i] = TVector<T>(size - i,
i);
}
template <typename T>
TMatrix<T>::TMatrix(const TMatrix<T>& matr) : TVector<TVector<T>>(matr)
{}
template <typename T>
TMatrix<T>::TMatrix(const       TVector<TVector<T>>&       vecs)       :
TVector<TVector<T>>(vecs) {}
template <typename T>
const TMatrix<T>& TMatrix<T>::operator = (const TMatrix<T>& matr) {
      return TVector<TVector<T>>::operator = (matr);
}
template <typename T>
int TMatrix<T>::operator == (const TMatrix<T>& matr) const {
      return TVector<TVector<T>>::operator == (matr);
}
template <typename T>
int TMatrix<T>::operator != (const TMatrix<T>& matr) const {
      return TVector<TVector<T>>::operator != (matr);
}
template <typename T>
TMatrix<T> TMatrix<T>::operator + (const TMatrix<T>& matr) {
      return TVector<TVector<T>>::operator + (matr);
}
template <typename T>
TMatrix<T> TMatrix<T>::operator - (const TMatrix<T>& matr) {
      return TVector<TVector<T>>::operator - (matr);
}
template <typename T>
TMatrix<T> TMatrix<T>::operator * (const TMatrix<T>& matr) {
      if (matr.size != size) throw "different_sizes";
      TMatrix<T> res(size), tmp(matr);
      for (int i = 0; i < size; i++) {
            for (int j = i; j < size; j++) res[i][j] = 0;
      }
      for (int i = 0; i < size; i++) {
            for (int j = i; j < size; j++) {
                  for   (int   k   =   i;   k   <=   j;   k++)   res[i][j]   +=
(*this)[i][k] * tmp[k][j];
            }
      }
      return res;
}
```