МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н. И. ЛОБАЧЕВСКОГО»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

Отчёт по лабораторной работе

**Классы для работы с векторами и матрицами**

Выполнила:

студентка ф-та ИТММ ПМИ – 381903-3

Исхакова Лана Руслановна

Проверил:

ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc24628114)

[Постановка задачи 4](#_Toc24628115)

[Руководство пользователя 5](#_Toc24628116)

[Руководство программиста 6](#_Toc24628117)-10

[Эксперименты 11-1](#_Toc24628118)3

[Заключение 1](#_Toc24628119)4

[Литература 1](#_Toc24628120)5

[Приложение 16-2](#_Toc24628121)7

# 1.Введение

Число является одним из основных понятий математики. Понятие числа развивалось в тесной связи с изучением величин; эта связь сохраняется и теперь. Во всех разделах современной математики и информатики приходится рассматривать разные величины и пользоваться числами.

В этой лабораторной работе мы столкнемся с такими понятиями, как вектор и матрица.

**Вектор** - направленный отрезок прямой, то есть отрезок, для которого указано, какая из его граничных точек является началом, а какая — концом. **Матрица** - [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82), записываемый в виде прямоугольной таблицы элементов [кольца](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) или [поля](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) (например, [целых](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), [действительных](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) или [комплексных](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) чисел), которая представляет собой совокупность [строк](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1) и [столбцов](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%BE%D0%BB%D0%B1%D0%B5%D1%86_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1), на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы.

Для матрицы определены следующие [алгебраические](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) операции:

* [сложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) матриц, имеющих один и тот же размер;
* [умножение матриц](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86) подходящего размера (матрицу, имеющую n {\displaystyle n} столбцов, можно умножить справа на матрицу, имеющую n {\displaystyle n} n строк);
* в том числе умножение на матрицу вектора (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);
* умножение матрицы на элемент основного кольца или поля (то есть [скаляр](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80)).

Для того чтобы работать с векторами и матрицами в программировании необходимы классы. **Класс** — это тип структуры, позволяющий включать в описание типа не только элементы данных, но и функции (функции-элементы или методы).

Также нам понадобятся шаблоны функций и шаблоны классов**. Шаблоны функций** – это обобщенное описание поведения функций, которые могут вызываться для объектов разных типов. **Шаблоны классов** – обобщенное описание пользовательского типа, в котором могут быть параметризованы атрибуты и операции типа.

# 

# 2.Постановка задачи

1. Написать классы для работы с векторами и матрицами использовать шаблоны.

2. Продемонстрировать их работу на примере (написать в main пример).

Должны быть:

* конструкторы (по умолчанию, инициализатор, копирования).
* деструктор.
* доступ к защищенным полям.
* перегруженные операции: +, -, \*, /, =, ==, [].
* потоковый ввод и вывод.
* перегруженные операции +, -, \*, / должны быть реализованы для векторов (вектор +, -, \*, / вектор), матриц (матрица +, -, \* матрица), матрично-векторные (матрица \* вектор и наоборот).
* в классе вектор должна быть возможность отсортировать его тремя способами.

3. Сравнить время работы, и сделать выводы.

# 3.Руководство пользователя

1. Запустить консоль программы «matrix.exe».
2. Создать объект типа vector или matrix выбирая между основными типами данных.
3. Произвести все необходимые операции над вектором или матрицей.
4. Чтобы использовать какую-нибудь функцию класса нужно обращаться к объекту класса через точку.

# 

# 4.Руководство программиста

***Описание структуры программы***

Программа состоит из одного решения, которое называется «matrix».

В решении содержится 2 элемента: «matrix.hpp», «main.cpp».

* В «matrix.hpp» определен класс Matrix и класс Vector, а также объявлены все их методы и их определения.
* В «main.cpp» определена стандартная функция int main.

***Описание структуры программы***

В программе определены шаблонные классы Vector и Matrix.

Внутри класса Matrix определены следующие поля (private):

* T\*\* data – шаблонный двойной указатель
* int m – количество строк
* int n – количество столбцов

Внутри класса matrix определены следующие поля(public):

* get\_n -метод, возвращающий количество строк.
* get\_m -метод, возвращающий количество столбцов.
* get\_lenght()– метод, устанавливающий размерности матрицы
* Matrix(int N, int M) – конструктор инциализатор, принимает на вход два параметра типа int, создает динамический массив заданного размера, заполняет его случайными числами.
* Matrix(Matrix &other):Matrix(other.n,other.m) – конструктор копирования, принимает на вход объект типа matrix, создает объект с теми же характеристиками, что и переданный, и матрицей того же содержимого

|  |
| --- |
|  |
|  |
|  |

* Matrix(Vector<T> &vec):Matrix(vec.get\_lenght(),1) конструктор от вектора, заполняет матрицу векторами
* ~Matrix() – деструктор
* Matrix operator +(Matrix & other) - перегрузка оператора суммы
* Matrix operator –(Matrix& other) - перегрузка оператора ­разности
* Matrix operator \*(Matrix& other) - перегрузка оператора умножения
* T\* operator [](int idx) - перегрузка оператора индексации
* Matrix &operator =(Matrix const &other) - перегрузка оператора равенства
* bool operator ==(Matrix &other) - перегрузка оператора сравнения
* friend std::istream &operator>>(std::istream &in, Matrix<B> &M) - перегрузка оператора вывода, позволяет выводить матрицы на экран
* friend std::ostream &operator<<(std::ostream &out, Matrix<B> &M) - перегрузка оператора ввода.
* int count\_num(T val) – метод, ищущий количества вхождений указанного значения

Две шаблонные функции:

* Matrix<T> operator\*(Matrix<T> M, Vector<T> V) – перегрузка оператора умножить, позволяет перемножать объекты типа Matrix с объектами типа Vector;
* Matrix<T> operator\*(Vector<T> V, Matrix<T> M) –– перегрузка оператора умножить, позволяет перемножать объекты типа Vector с объектами типа Matrix;

Внутри класса Vector определены следующие поля(private):

* T \*data – шаблонный указатель;
* int length – количество строк вектора.

Внутри класса Vector определены следующие поля(public):

* ~Vector() – деструктор
* Vector(int sz = 0) : lenght(sz) – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля
* Vector(Vector &other):Vector(other.lenght) - – конструктор копирования, принимает на вход объект типа Vector, создает объект с теми же характеристиками, что и переданный
* get\_lenght()– метод, возвращающий размерность вектора
* Vector operator +( Vector const &other) - перегрузка оператора суммы
* Vector operator –( Vector const &other) - перегрузка оператора ­разности
* Vector operator \*( Vector const &other) - перегрузка оператора умножения
* Vector operator =( Vector const &other) - перегрузка оператора равенства
* bool operator ==( Vector const &other) - перегрузка оператора сравнения
* friend std::ostream &operator<<(std::ostream &out, Vector<T> &V) - перегрузка оператора вывода, позволяет выводить матрицы на экран
* friend std::istream &operator>>(std::istream &in, Vector<T> &V) - перегрузка оператора ввода
* T& operator[](int i) - перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу
* Vector operator/(Vector const &other) – перегрузка оператора деления возвращает вектор, координаты которого являются частным от деления двух соответствующих координат операндов
* void sort\_bubble() - сортировка «Пузырек».
* void sort\_insertiont() - сортировка «Вставка».
* void sort\_quick() - сортировка «Быстрая».

***Описание алгоритмов***

1. **Сортировка пузырьком**

Главной идеей алгоритма является то, что сравниваются два соседних элемента, и если они стоят в неправильном порядке, то эти элементы меняются местами. Таким образом, за каждый проход в конце массива встает наибольшее число, если сортируется по возрастанию, или наименьшее, если по убыванию. Благодаря этому, программа может не проверят его на следующей итерации.

1. **Сортировка вставкой**

На вход алгоритма подаётся последовательность n чисел: a1, а2, ..., аn . Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a’1, а’2, …, а’n , чтобы выполнялось следующее соотношение a’1 ≤ а’2 ≤ … ≤а’n .

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

1. **Быстрая сортировка**

Общая идея алгоритма состоит в следующем:

• Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

• Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

• Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

# 5.Эксперименты

# Эксперименты

Оценим время, которое занимают матричные, векторно-матричные операции, с помощью асимптотической сложности.

1. Рассмотрим код отвечающий за суммирование матриц:

Matrix operator+(Matrix &other)

{

Matrix res(n,m);

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

for(int j = 0;j<m;++j)

res.data[i][j]=data[i][j]+other.data[i][j];

return res;

}

Асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.62 |
| 2000 | 2.15 |
| 4000 | 8.42 |

Таблица 1: Время суммирования матриц.

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

1. Теперь аналогично оценим асимптотическую сложность матричного умножения:

Matrix operator\*(Matrix &other)

{

Matrix res(n,other.m);

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

for(int j = 0;j<other.m;++j)

for(int k = 0;k<m;++k)

res.data[i][j] += data[i][k]\*other.data[k][j];

return res;

}

Асимптотическая сложность умножения:

Теперь мы произведем замеры времени умножения матриц

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 9.40 |
| 2000 | 79.37 |
| 4000 | 6034.6 |

Таблица 2: Время умножения матриц.

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 8 раза.

1. Теперь аналогично оценим асимптотическую сложность векторно-матричного умножения:

|  |
| --- |
|  |
|  |
|  | Matrix<T> operator\*(Matrix<T> M, Vector<T> V)  { |
|  | Matrix<T> buf(V); |
|  | return M\*buf; |
|  | } |
|  |  |

Асимптотическая сложность этого кода :

Теперь мы произведем замеры времени умножения матриц и векторов

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000 | 0.002 |
| 2000 | 0.014 |
| 4000 | 0.053 |

Таблица 3: Время умножения матриц и векторов

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

Теперь перейдем к сортировкам. Асимптотическая сложность уже определена. Поэтому просто произведём замеры и составим таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сортировка пузырьком | Сортировка вставками | Быстрая сортировка  (Хоара) |
| 100 000 элементов | 14.98 сек | 1.20 сек | 0.013 сек |
| 200 000 элементов | 62.16 сек | 7.28 сек | 0.025 сек |
| 400 000 элементов | 253.30 сек | 29.20 сек | 0.057 сек |
| Асимптотическая сложность |  |  |  |

Таблица 4: Время работы сортировок

Как видно из таблицы, действительно, затрачиваемое на реализацию операции время растет эквивалентно асимптотической функции.

# 6.Заключение

Мной была написана программа, позволяющая работать с векторами и матрицами. В частности выполнять все алгебраические операции с использованием шаблонов.

Эксперименты показали, что то время, которое затрачивается в теории на реализацию матричных, матрично-векторных операций с некоторыми погрешностями совпадает с асимптотической сложности времени, которого мы ожидали.

# 

# 7.Литература

1. С. Н. Марков. Курс истории математики: Учебное пособие. - Иркутск: Издательство иркутского университета, 1995. 248 с.
2. <http://www.math24.ru/множества-чисел.html>
3. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003
4. <https://ru.wikipedia.org>

# 

# 8.Приложение

***Приложение 1. Код программы***

**main.cpp**

|  |
| --- |
| #include <cmath> |
|  | #include <iostream> |
|  | #include "Matrix.hpp" |
|  |  |
|  | int main() |
|  | { |
|  | Matrix<int> a,b,c; |
|  | std::cout<<"Ener matrix (n, m, mat): "; |
|  | std::cin>>a; |
|  | b=a; |
|  | std::cout<<"b=a, a==b = "<<static\_cast<int>(a==b)<<"\n"; |
|  | c = a+b; |
|  | std::cout<<"a + b = \n"<<c<<"\n"; |
|  | c = a - b; |
|  | std::cout<<"a - b = \n"<<c<<"\n"; |
|  | c = a\*b; |
|  | std::cout<<"a \* b = \n"<<c<<"\n"; |
|  | std::cout<<"a[1][1] = "<<a[1][1]<<"\n"; |
|  |  |
|  | Vector<int> d,e,f; |
|  | std::cout<<"Ener vector (n, vec): "; |
|  | std::cin>>d; |
|  | e=d; |
|  | std::cout<<"b=a, a==b = "<<static\_cast<int>(d==e)<<"\n"; |
|  | d.sort\_quick(0,d.get\_lenght()-1); |
|  | std::cout<<"a.sort() \n"<<d<<"\n"; |
|  | f = d+e; |
|  | std::cout<<"a + b = \n"<<f<<"\n"; |
|  | f = d - e; |
|  | std::cout<<"a - b = \n"<<f<<"\n"; |
|  | f = d\*e; |
|  | std::cout<<"a \* b = \n"<<f<<"\n"; |
|  | f = d/e; |
|  | std::cout<<"a / b = \n"<<f<<"\n"; |
|  | std::cout<<"a[1] = "<<d[1]<<"\n"; |
|  | std::cout<<"Enter matrix and vec "; |
|  | std::cin>>a>>d; |
|  | auto l = a\*d; |
|  | std::cout<<"Mat\*vec = "<<l<<"\n"; |
|  |  |
|  | return 0; |
|  | } |

**matrix.hpp**

|  |
| --- |
| #pragma once |
|  | #include <iostream> |
|  |  |
|  | template <class T> |
|  | class Vector |
|  | { |
|  | public: |
|  | Vector(int sz = 0) : lenght(sz) |
|  | { |
|  | data = new T[sz]; |
|  | for(int i = 0;i<sz;++i) |
|  | data[i]=0; |
|  | } |
|  |  |
|  | Vector(Vector &other):Vector(other.lenght) |
|  | { |
|  | for(int i = 0;i<lenght;++i) |
|  | data[i]=other.data[i]; |
|  | } |
|  |  |
|  | Vector &operator=(Vector const &other) |
|  | { |
|  | if (&other == this) |
|  | return \*this; |
|  |  |
|  | delete[] data; |
|  | data = new T[other.lenght]; |
|  | lenght = other.lenght; |
|  | for (int i = 0; i < lenght; ++i) |
|  | data[i] = other.data[i]; |
|  | return \*this; |
|  | } |
|  |  |
|  | Vector operator+(Vector const &other) |
|  | { |
|  | Vector res(lenght); |
|  | for(int i = 0;i<lenght;++i) |
|  | res.data[i]=data[i]+other.data[i]; |
|  | return res; |
|  | } |
|  |  |
|  | Vector operator-(Vector const &other) |
|  | { |
|  | Vector res(lenght); |
|  | for(int i = 0;i<lenght;++i) |
|  | res.data[i]=data[i]-other.data[i]; |
|  | return res; |
|  | } |
|  |  |
|  | Vector operator\*(Vector const &other) |
|  | { |
|  | Vector res(lenght); |
|  | for(int i = 0;i<lenght;++i) |
|  | res.data[i]=data[i]\*other.data[i]; |
|  | return res; |
|  | } |
|  |  |
|  | Vector operator/(Vector const &other) |
|  | { |
|  | Vector res(lenght); |
|  | for(int i = 0;i<lenght;++i) |
|  | res.data[i]=data[i]/other.data[i]; |
|  | return res; |
|  | } |
|  |  |
|  | bool operator==(Vector &other) |
|  | { |
|  | if(lenght!=other.lenght) |
|  | return 0; |
|  | for(int i = 0;i<lenght;++i) |
|  | if(data[i]!=other.data[i]) |
|  | return 0; |
|  | return 1; |
|  | } |
|  |  |
|  | T& operator[](int i) |
|  | { |
|  | return data[i]; |
|  | } |
|  |  |
|  | T\* get\_data() |
|  | { |
|  | return data; |
|  | } |
|  |  |
|  | int get\_lenght() |
|  | { |
|  | return lenght; |
|  | } |
|  |  |
|  | void sort\_bubble() |
|  | { |
|  | for(int i = 0;i<lenght-1;++i) |
|  | for(int j = 0;j<lenght-1;++j) |
|  | if(data[j]>data[j+1]) |
|  | swap(data[j],data[j+1]); |
|  | } |
|  |  |
|  | void sort\_insertion() |
|  | { |
|  | for(int i = 1;i<lenght;++i) |
|  | { |
|  | int j = i-1; |
|  | while(j>=0&&data[j]>data[j+1]) |
|  | { |
|  | swap(data[j],data[j+1]); |
|  | --j; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int part(int l, int r) |
|  | { |
|  | T v = data[(l+r)/2]; |
|  | int i = l; |
|  | int j = r; |
|  | while(i<=j) |
|  | { |
|  | while(data[i]<v) |
|  | ++i; |
|  | while(data[j]>v) |
|  | --j; |
|  | if(i>=j) |
|  | break; |
|  | swap(data[i++],data[j--]); |
|  | } |
|  | return j; |
|  | } |
|  |  |
|  | void sort\_quick(int l , int r) |
|  | { |
|  | if(l<r) |
|  | { |
|  | int q = part(l,r); |
|  | sort\_quick(l,q); |
|  | sort\_quick(q+1,r); |
|  | } |
|  | } |
|  |  |
|  | ~Vector() |
|  | { |
|  | delete[] data; |
|  | } |
|  |  |
|  | template <class B> friend std::istream &operator>>(std::istream &in, Vector<B> &V); |
|  | template <class B> friend std::ostream &operator<<(std::ostream &out, Vector<B> &V); |
|  |  |
|  | private: |
|  | T \*data; |
|  | int lenght; |
|  | }; |
|  |  |
|  | template <class T> |
|  | std::istream &operator>>(std::istream &in, Vector<T> &V) |
|  | { |
|  | T buf; |
|  | int len; |
|  | in >> len; |
|  | V = Vector<T>(len); |
|  | for (int i = 0; i < len; ++i) |
|  | in >> V.data[i]; |
|  | return in; |
|  | } |
|  |  |
|  | template <class T> |
|  | std::ostream &operator<<(std::ostream &out, Vector<T> &V) |
|  | { |
|  | for (int i = 0; i < V.lenght; ++i) |
|  | out << V.data[i]<<" "; |
|  | return out; |
|  | } |
|  |  |
|  | template <class T> |
|  | class Matrix |
|  | { |
|  | public: |
|  | Matrix(int N = 0, int M = 0):n(N),m(M) |
|  | { |
|  | data = new T\*[n]; |
|  | for(int i = 0;i<n;++i) |
|  | data[i] = new T[m]; |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | data[i][j]=0; |
|  | } |
|  |  |
|  | Matrix(Matrix &other):Matrix(other.n,other.m) |
|  | { |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | data[i][j]=other.data[i][j]; |
|  | } |
|  |  |
|  | Matrix(Vector<T> &vec):Matrix(vec.get\_lenght(),1) |
|  | { |
|  | for(int i = 0;i<n;++i) |
|  | data[i][0]=vec[i]; |
|  | } |
|  |  |
|  | Matrix &operator=(Matrix const &other) |
|  | { |
|  | if (&other == this) |
|  | return \*this; |
|  |  |
|  | for(int i = 0;i<n;++i) |
|  | delete[] data[i]; |
|  | delete[] data; |
|  |  |
|  | n = other.n; |
|  | m=other.m; |
|  | data = new T\*[n]; |
|  | for(int i = 0;i<n;++i) |
|  | data[i] = new T[m]; |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | data[i][j]=other.data[i][j]; |
|  | return \*this; |
|  | } |
|  |  |
|  | Matrix operator\*(Matrix &other) |
|  | { |
|  | Matrix res(n,other.m); |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<other.m;++j) |
|  | for(int k = 0;k<m;++k) |
|  | res.data[i][j] += data[i][k]\*other.data[k][j]; |
|  | return res; |
|  | } |
|  |  |
|  | Matrix operator+(Matrix &other) |
|  | { |
|  | Matrix res(n,m); |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | res.data[i][j]=data[i][j]+other.data[i][j]; |
|  | return res; |
|  | } |
|  |  |
|  | Matrix operator-(Matrix &other) |
|  | { |
|  | Matrix res(n,m); |
|  | for(int i = 0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | res.data[i][j]=data[i][j]-other.data[i][j]; |
|  | return res; |
|  | } |
|  |  |
|  | bool operator==(Matrix &other) |
|  | { |
|  | if(n!=other.n) |
|  | return 0; |
|  | if(m!= other.m) |
|  | return 0; |
|  | for(int i =0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | if(data[i][j]!=other.data[i][j]) |
|  | return 0; |
|  | return 1; |
|  | } |
|  |  |
|  | T\* operator[](int idx) |
|  | { |
|  | return data[idx]; |
|  | } |
|  |  |
|  | int get\_n() |
|  | { |
|  | return n; |
|  | } |
|  |  |
|  | int get\_m() |
|  | { |
|  | return m; |
|  | } |
|  |  |
|  | int count\_num(T val) |
|  | { |
|  | int res=0; |
|  | for(int i = 0;i<n;++i) |
|  | for(int j=0;j<m;++j) |
|  | if(data[i][j]==val) |
|  | ++res; |
|  | return res; |
|  | } |
|  |  |
|  | template <class B> friend std::istream &operator>>(std::istream &in, Matrix<B> &M); |
|  | template <class B> friend std::ostream &operator<<(std::ostream &out, Matrix<B> &M); |
|  |  |
|  | private: |
|  | T\*\* data; |
|  | int n; |
|  | int m; |
|  | }; |
|  |  |
|  | template <class T> |
|  | std::istream &operator>>(std::istream &in, Matrix<T> &M) |
|  | { |
|  | int n,m; |
|  | in >> n>>m; |
|  | M = Matrix<T>(n,m); |
|  | for(int i =0;i<n;++i) |
|  | for(int j = 0;j<m;++j) |
|  | in>>M.data[i][j]; |
|  | return in; |
|  | } |
|  |  |
|  | template <class T> |
|  | std::ostream &operator<<(std::ostream &out, Matrix<T> &M) |
|  | { |
|  | for(int i = 0;i<M.n;++i) |
|  | { |
|  | for(int j = 0;j<M.m;++j) |
|  | std::cout<<M.data[i][j]<<" "; |
|  | std::cout<<"\n"; |
|  | } |
|  | return out; |
|  | } |
|  |  |
|  | template <class T> |
|  | Matrix<T> operator\*(Matrix<T> M, Vector<T> V) |
|  | { |
|  | Matrix<T> buf(V); |
|  | return M\*buf; |
|  | } |
|  |  |
|  | template <class T> |
|  | Matrix<T> operator\*(Vector<T> V, Matrix<T> M) |
|  | { |
|  | Matrix<T> buf(V); |
|  | return buf\*M; |
|  | } |