Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И. Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Статические библиотеки и шаблоны.

Выполнил:

студент группы 3821Б1ПМ3

Одинец В.В.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И. Г.

Нижний Новгород

2022г.

Содержание

1. Введение 2

2. Постановка задачи 3

3. Руководство пользователя 4

4. Руководство программиста 7

*4.1 Описание структуры программы* 7

*4.2 Описание структуры данных* 15

*4.3 Описание алгоритмов* 16

5. Эксперименты 17

6. Заключение 18

7. Литература 19

8. Приложения 20

*8.1* *Приложение 1* 20

*8.1* *Приложение 2* 24

**1. Введение.**

Целью данной лабораторной работы является ознакомление с разработкой статических библиотек для работы с векторами и матрицами. А также научиться работать шаблонами, которые позволяют написать реализацию сразу нескольких типов данных, что упрощает разработку, когда не надо для каждого типа писать отдельную реализацию.

**2. Постановка задачи.**

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

Вектора в математическом понимании: имеется набор значений из N мерного пространства, размерность задается как параметр.

Матрица должна быть наследником вектора.

Классы вектора и матрицы должны быть вынесены в статическую библиотеку.

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)

**3. Руководство пользователя.**

Создание векторов. Для работы с векторами для начала надо уметь их создавать. Для этого в программе реализованы сразу 4 конструктора векторов: по умолчанию, 2 конструктора инициализации и конструктор копирования.

Для конструктора по умолчанию нужно лишь написать тип данных вектор, указать тип данных и написать имя вектора. Для первого конструктора инициализации надо внутри вектора написать его длину и тогда получится вектор введённой длины и проинициализированный нулями. Для второго конструктора инициализации надо внутри помимо длины написать элемент, которым заполнить все координаты вектора. Для конструктора копирования надо внутри ввести вектор, с которого скопируются длина и координаты.

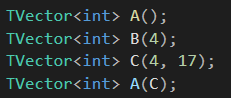


Рис 1. Примеры создания вектора в программе

После создания вектора его значения можно изменить при помощи вызова операции ввода («<<»). В таком случае программа будет запрашивать у пользователя значения координат вектора по порядку. (Рис. 2)

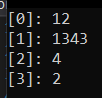


Рис 2. Ввод значений вектора длины 4

Над векторами можно выполнять различные арифметические операции, такие как +, -, \*, /, =, ==, [], а также потоковый ввод и вывод. (Рис. 3)

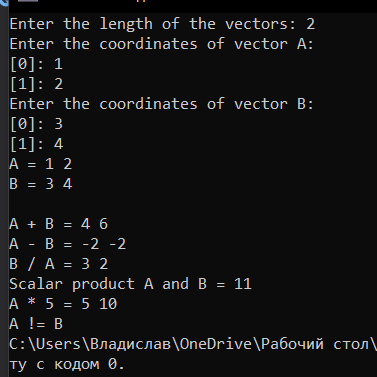


Рис 3. Введение значений векторов и примеры арифметических операций над ними.

Создание матриц. Для создания и работы с матрица есть 5 конструкторов: по умолчанию, 3 конструктора - инициализатора, копирования. Конструктор по умолчанию создает пустую матрицу. Первый конструктор инициализации получает одно значение типа int, которое обозначает количество строк. После этого создаётся матриц 1 на введённое значение. Второй конструктор инициализации получает два значения типа int, где первое обозначает количество строк(n), а другое количество символов(m). После этого создаётся размерами n на m. Третий конструктор инициализации получает два значения типа int и одно значение шаблонного типа. После этого создаётся размерами n на m, заполненная третьим значением. Конструктор копирования получает значения типа TMatrix и копирует его значения полностью.

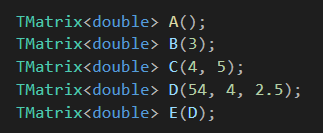


Рис 4. Примеры создания матрицы в программе.

После создания матрицы её значения можно изменить при помощи вызова операции ввода («<<»). В таком случае программа будет запрашивать у пользователя значения матрицы по порядку(слева направо, снизу вверх). (Рис. 5)

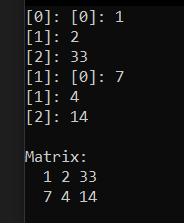


Рис 5. Ввод значений в матрицу размеров 2 на 3

Над матрицами можно выполнять различные арифметические операции, такие как +, -, \*, /, =, ==, [], а также потоковый ввод и вывод. (Рис. 6)

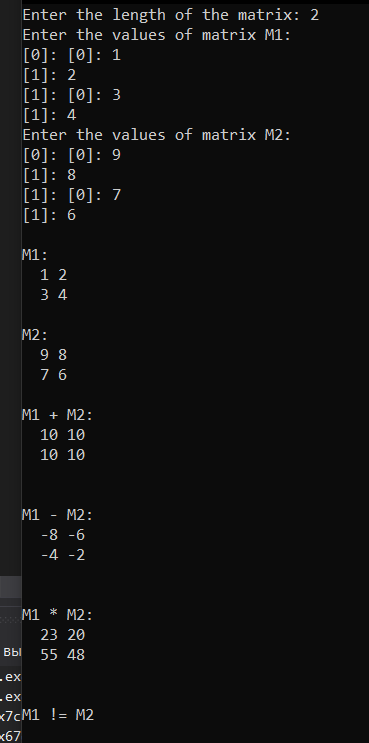


Рис 6. Введение значений матриц и примеры арифметических операций над ними.

**4. Руководство программиста.**

***4.1 Описание структуры программы.***

**4.1.1 Класс TVector**

**1) Объявление шаблонного класса**

Объявляем класс векторов с полями «data» и «len». Поле «data» отвечает за хранение координат вектора, а «len» за хранение длины этого вектора.

Фрагмент кода 1:

template<class T>

class TVector {

protected:

T\* data;

int len;

public:

TVector();

TVector(int n);

TVector(int n, T v);

TVector(const TVector<T>& p);

~TVector();

int GetLen() const;

T& operator[](int i);

TVector<T>& operator = (const TVector<T>& p);

TVector<T> operator + (const TVector<T>& p);

TVector<T> operator - (const TVector<T>& p);

TVector<T> operator / (const TVector<T>& p);

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

TVector<T> operator \* (const T n);

bool operator == (const TVector<T>& p);

friend ostream& operator<<(ostream& ostr, TVector<T>& p)

{

for (int i = 0; i < p.GetLen(); i++)

ostr << " " << p[i] ;

if (int j = 0 <= p.GetLen()) {

cout << "\n";

j++;

}

else

{

cout << "\t ";

++j;

}

return ostr;

}

friend istream& operator>>(istream& t, TVector<T>& v)

{

if (v.GetLen() == 0)

throw "length must be > 0";

for (int i = 0; i < v.GetLen(); i++) {

cout << "[" << i << "]: ";

t >> v[i];

}

return t;

}

};

**2) Реализация конструктора и деструктора**

В реализации имеется 4 конструктора: по умолчанию, 2 конструктора инициализации и конструктор копирования, а также деструктор.

Фрагмент кода 2:

template<class T>

inline TVector<T>::TVector()

{

len = 0;

data = nullptr;

}

template<class T>

inline TVector<T>::TVector(int n, T v)

{

if (n > 0)

{

data = new T[n];

len = n;

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

data[i] = v;

}

else

{

cout << "length <= 0"<< endl;

}

}

template<class T>

inline TVector<T>::TVector(int n)

{

data = new T[n];

len = n;

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

data[i] = 0;

}

template<class T>

inline TVector<T>::TVector(const TVector<T>& p)

{

data = new T[p.len];

len = p.len;

for (int i = 0; i < p.len; i++)

data[i] = p.data[i];

}

template<class T>

inline TVector<T>::~TVector()

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

}

**3) Реализация методов и операторов**

Фрагмент кода 3:

template<class T>

inline int TVector<T>::GetLen() const

{

return len;

}

template<class T>

inline T& TVector<T>::operator[](int i)

{

if (i >= 0 && i < len)

{

return data[i];

}

else

{

cout << "out of range";

}

}

template<class T>

inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

data = new T[p.len];

len = p.len;

for (int i = 0; i < p.len; i++)

data[i] = p.data[i];

return \*this;

}

template<class T>

inline TVector<T> TVector<T>::operator+(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) + p.data[q];

}

return res;

}

}

template<class T>

inline TVector<T> TVector<T>::operator-(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) - p.data[q];

}

return res;

}

}

template<class T>

inline TVector<T> TVector<T>::operator/(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) / p.data[q];

}

return res;

}

}

template<class T>

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

{

if (len != v.GetLen()) {

cout << "different sizes";

}

T res = 0;

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

res += (this->data[i]) \* v.data[i];

}

return res;

}

template<class T>

inline TVector<T> TVector<T>::operator\*(const T n)

{

TVector<T> res(len);

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

res[i] = data[i] \* n;

return res;

}

template<class T>

inline bool TVector<T>::operator==(const TVector<T>& p)

{

if (len != p.len)

return false;

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

if (data[i] != p.data[i])

return false;

return true;

}

**4.1.2 Класс TMatrix**

**1) Объявление шаблонного класса**

Объявляем класс матриц. Он является наследником класса векторов. Так как он наследник класса векторов, то в него передадутся поля из класса векторов. Объявляем поле «len1», которое хранит количество строк.

Фрагмент кода 4:

template<class T>

class TMatrix : public TVector<TVector<T>>

{

protected:

int len1;

public:

TMatrix();

TMatrix(int n1);

TMatrix(int n1, int n2);

TMatrix(int n1, int n2, T v);

TMatrix(const TMatrix<T>& m);

~TMatrix();

int GetLen1() const;

TMatrix<T>& operator = (const TMatrix<T>& m);

bool operator == (const TMatrix<T>& m);

TMatrix<T> operator + (const TMatrix<T>& m);

TMatrix<T> operator - (const TMatrix<T>& m);

TMatrix<T> operator \* (const TMatrix<T>& p);

TMatrix<T> operator \* (const T c);

};

**2) Реализация конструктора и деструктора**

В реализации имеется 5 конструктора: по умолчанию, 3 конструктора инициализации и конструктор копирования, а также деструктор.

Фрагмент код 5:

template<class T>

inline TMatrix<T>::TMatrix() : TVector<TVector<T>>::TVector()

{

this ->len = 0;

this-> len1 = 0;

this-> data = 0;

}

template<class T>

inline TMatrix<T>::TMatrix(int n1) : TVector<TVector<T>>::TVector(n1)

{

len1 = 1;

for (int i = 0; i < n1; i++)

{

this->data[i] = 1;

for (int j = 0; j < 1; j++) {

data[i][j] = 0;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(int n1, int n2) : TVector<TVector<T>>::TVector(n1)

{

len1 = n2;

for (int i = 0; i < n1; i++)

{

this->data[i] = n2;

for (int j = 0; j < n2; j++) {

data[i][j] = 0;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(int n1, int n2, T v) : TVector<TVector<T>>::TVector(n1)

{

len1 = n2;

for (int i = 0; i < n1; i++)

{

this->data[i] = n2;

for (int j = 0; j < n2; j++) {

data[i][j] = v;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(const TMatrix<T>& m)

{

len1 = m.len1;

this->len = m.len;

this->data = new TVector<T>[len1];

for (int i = 0; i < len1; i++)

this->data[i] = TVector<T>(this->len);

this->len = m.len;

for (int i = 0; i < m.len1; i++)

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

this->data[i][j] = m.data[i][j];

}

template<class T>

inline TMatrix<T>::~TMatrix()

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

}

**3) Реализация методов и операторов**

Фрагмент кода 6:

template<class T>

inline int TMatrix<T>::GetLen1() const

{

return len1;

}

template<class T>

inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& m)

{

if (this == &m) return \*this;

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;}

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

data[i][j] = m.data[i][j];

return \*this;

}

template<class T>

inline bool TMatrix<T>::operator==(const TMatrix<T>& m)

{

if (this->len != m.len || len1 != m.len1) return false;

for (int i = 0; i < len1; i++)

{

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

if (data[i][j] != m.data[i][j]) return false;

}

return true;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator+(const TMatrix& m)

{

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;

}

TMatrix res(\*this);

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

res.data[i][j] = this->data[i][j] + m.data[i][j];

return res;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator-(const TMatrix& m)

{

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;

}

TMatrix res(\*this);

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

res.data[i][j] = this->data[i][j] - m.data[i][j];

return res;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)

{

TMatrix<T> res(p.len, len1, 0);

for (int i = 0; i < len1; i++)

{

for (int j = 0; j < p.len; j++)

{

res[i][j] = 0;

for (int l = 0; l < this->len; l++)

res[i][j] += this->data[i][l] \* p.data[l][j];

}

}

return res;

}

template <class T>

TMatrix<T> TMatrix<T>::operator\*(const T a)

{

TMatrix<T> res(\*this);

for (int i = 0; i < len1; i++) {

res[i] = res[i] \* a;

}

return res;

}

***4.2 Описание структуры данных***

**1) TVector**

TVector имеет два модификатора доступа: «public» и «protected». В public находятся все конструкторы, деструкторы, методы и операции. В protected находятся защищённые поля «data» и «len». «Data» является массивом, в котором хранятся значение координат вектора. «Len» имеет тип данных int и хранит в себе длину вектора.

TVector является шаблонным классом, что означает, что класс может работать с разными типами данных.

**2) TMatrix**

Класс TMatrix является наследником класса TVector с ключом доступа «public», что означает, что в TMatrix перешли все поля и методы из TVector. В «TMatrix» было объявлено новое поле «len1», которое имеет тип данных int и хранит в себе количество строк.

***4.3 Описание алгоритмов***

TVector. Для операций векторов +, -, / есть операция вычисления вектора c, все элементы которого равны попарной сумме, разности или делению соответствующих элементов векторов a и b, то есть каждый элемент вектора c равен:, где это +, -, /.

Умножение векторов в данной программе является скалярным произведением:

TMatrix. Так же, как и у векторов складываются соответствующие значения, но у матрицы помимо номера элемента, есть ещё номер строки, в которой он находится, то есть каждый элемент вектора c равен:, где это +, -, /.

Произведением двух матриц А и В называется матрица С, элемент которой, находящийся на пересечении i-й строки и j-го столбца, равен сумме произведений элементов i-й строки матрицы А на соответствующие (по порядку) элементы j-го столбца матрицы В. Из этого определения следует формула элемента матрицы C:

сij = ai1b1j + ai1b1j + … + aipbpj

**5. Эксперименты.**

***5.1 Сравнение времени операции сложения с матрицами.***

Для измерения времени будем использовать библиотек «time.h»

Пример измерения времени.

Фрагмент кода 7:

clock\_t t1 = clock();

TMatrix<double> A(1000, 1000, 1);

TMatrix<double> B(1000, 1000, 2);

TMatrix<double> C(1000, 1000);

C = A + B;

clock\_t t2 = clock();

cout << "seconds = " << (double)(t2 - t1) / CLOCKS\_PER\_SEC << endl;

Проведём первый замер суммы для матрицы 1000 x 1000. Получили время 0.025 м/с. Сложность алгоритма сложения o(n^2). Проведём замер для матрицы 10000 х 10000. Значит замер времени должен отличаться в 100 раз. Получили время 2.535 м/с. И так же проверим для матрицы 12000 х 12000. Время на ней должно отличаться примерно на 1.44 от времени матрицы 10000 х 10000. Время матрицы 12000 х 12000 равно 3.405.

|  |  |  |
| --- | --- | --- |
| Размерность | Время теоретическое | Время практическое |
| 1000 x 1000 | 0.025 | 0.025 |
| 10000 x 10000 | 2.5 | 2.535 |
| 12000 x 12000 | 3.6 | 3.405 |

Теоретическое время приблизительно совпадает с практическим.

***5.2 Сравнение времени операции сложения с матрицами.***

Сложность алгоритма сложения o(n^3). Проведём первый замер суммы для матрицы 100 x 100. Получили время 0.004 м/с. Проведём замер для матрицы 1000 х 1000. Время должен отличаться в 1000 раз. Получили время 4.355 м/с.

|  |  |  |
| --- | --- | --- |
| Размерность | Время теоретическое | Время практическое |
| 100 x 100 | 0.004 | 0.004 |
| 1000 x 1000 | 4 | 4.355 |

**6. Заключение.**

В ходя данной лабораторной работы мы написали классы для работы с векторами и матрицами, а также использовали внутри них шаблоны, которые позволяют работать им с различными типами данных.

**7. Литература.**

1) Т. А. Павловская. «C/C++ Программирование на языке высокого уровня». Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>

2) Бьерн Страуструп. «Язык программирования С++». Режим доступа: <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>

3) Б. В. Керниган, Д. М. Ричи. «ЯЗЫК С». Режим доступа: <https://nsu.ru/xmlui/bitstream/handle/nsu/9058/kr.pdf>

**8. Приложения.**

***8.1. Приложение 1***.

#pragma once

#include <iostream>

using namespace std;

template<class T>

class TVector

{

protected:

T\* data;

int len;

public:

TVector();

TVector(int n, T v);

TVector(int n);

TVector(const TVector<T>& p);

~TVector();

int GetLen() const;

T& operator[](int i);

TVector<T>& operator = (const TVector<T>& p);

TVector<T> operator + (const TVector<T>& p);

TVector<T> operator - (const TVector<T>& p);

TVector<T> operator / (const TVector<T>& p);

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

TVector<T> operator \* (const T n);

bool operator == (const TVector<T>& p);

friend ostream& operator<<(ostream& ostr, TVector<T>& p)

{

for (int i = 0; i < p.GetLen(); i++)

ostr << " " << p[i] ;

if (int j = 0 <= p.GetLen()) {

cout << "\n";

j++;

}

else

{

cout << "\t ";

++j;

}

return ostr;

}

friend istream& operator>>(istream& t, TVector<T>& v)

{

if (v.GetLen() == 0)

throw "length must be > 0";

for (int i = 0; i < v.GetLen(); i++) {

cout << "[" << i << "]: ";

t >> v[i];

}

return t;

}

};

template<class T>

inline TVector<T>::TVector()

{

len = 0;

data = nullptr;

}

template<class T>

inline TVector<T>::TVector(int n, T v)

{

if (n > 0)

{

data = new T[n];

len = n;

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

data[i] = v;

}

else

{

cout << "length <= 0"<< endl;

}

}

template<class T>

inline TVector<T>::TVector(int n)

{

data = new T[n];

len = n;

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

data[i] = 0;

}

template<class T>

inline TVector<T>::TVector(const TVector<T>& p)

{

data = new T[p.len];

len = p.len;

for (int i = 0; i < p.len; i++)

data[i] = p.data[i];

}

template<class T>

inline TVector<T>::~TVector()

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

}

template<class T>

inline int TVector<T>::GetLen() const

{

return len;

}

template<class T>

inline T& TVector<T>::operator[](int i)

{

return data[i];

}

template<class T>

inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

data = new T[p.len];

len = p.len;

for (int i = 0; i < p.len; i++)

data[i] = p.data[i];

return \*this;

}

template<class T>

inline TVector<T> TVector<T>::operator+(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) + p.data[q];

}

return res;

}

}

template<class T>

inline TVector<T> TVector<T>::operator-(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) - p.data[q];

}

return res;

}

}

template<class T>

inline TVector<T> TVector<T>::operator/(const TVector<T>& p)

{

if (this->len != p.len) {

cout << "sizes don't match" << endl;

}

else {

TVector<T> res(\*this);

for (int q = 0; q < len; q++)

{

res.data[q] = (this->data[q]) / p.data[q];

}

return res;

}

}

template<class T>

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

{

if (len != v.GetLen()) {

cout << "different sizes";

}

T res = 0;

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

res += (this->data[i]) \* v.data[i];

}

return res;

}

template<class T>

inline TVector<T> TVector<T>::operator\*(const T n)

{

TVector<T> res(len);

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

res[i] = data[i] \* n;

return res;

}

template<class T>

inline bool TVector<T>::operator==(const TVector<T>& p)

{

if (len != p.len)

return false;

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

if (data[i] != p.data[i])

return false;

return true;

}

***8.2. Приложение 2***.

#pragma once

#include "Vector.h"

template<class T>

class TMatrix : public TVector<TVector<T>>

{

protected:

int len1;

public:

TMatrix();

TMatrix(int n1);

TMatrix(int n1, int n2);

TMatrix(int n1, int n2, T v);

TMatrix(const TMatrix<T>& m);

~TMatrix();

int GetLen1() const;

TMatrix<T>& operator = (const TMatrix<T>& m);

bool operator == (const TMatrix<T>& m);

TMatrix<T> operator + (const TMatrix<T>& m);

TMatrix<T> operator - (const TMatrix<T>& m);

TMatrix<T> operator \* (const TMatrix<T>& p);

TMatrix<T> operator \* (const T c);

//TMatrix<T> operator \* (const TVector<T>& v);

};

template<class T>

inline TMatrix<T>::TMatrix() : TVector<TVector<T>>::TVector()

{

this ->len = 0;

this-> len1 = 0;

this-> data = 0;

}

template<class T>

inline TMatrix<T>::TMatrix(int n1) : TVector<TVector<T>>::TVector(n1)

{

len1 = 1;

for (int i = 0; i < n1; i++)

{

this->data[i] = 1;

for (int j = 0; j < 1; j++) {

data[i][j] = 0;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(int n1, int n2) : TVector<TVector<T>>::TVector(n1)

{

len1 = n2;

for (int i = 0; i < n1; i++)

{

this->data[i] = n2;

for (int j = 0; j < n2; j++) {

data[i][j] = 0;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(int n1, int n2, T v) : TVector<TVector<T>>::TVector(n1)

{

len1 = n2;

for (int i = 0; i < n1; i++)

{

this->data[i] = n2;

for (int j = 0; j < n2; j++) {

data[i][j] = v;

}

}

}

template<class T>

inline TMatrix<T>::TMatrix(const TMatrix<T>& m)

{

len1 = m.len1;

this->len = m.len;

this->data = new TVector<T>[len1];

for (int i = 0; i < len1; i++)

this->data[i] = TVector<T>(this->len);

this->len = m.len;

for (int i = 0; i < m.len1; i++)

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

this->data[i][j] = m.data[i][j];

}

template<class T>

inline TMatrix<T>::~TMatrix()

{

if (data != nullptr)

{

delete[] data;

data = nullptr;

}

}

template<class T>

inline int TMatrix<T>::GetLen1() const

{

return len1;

}

template<class T>

inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& m)

{

if (this == &m) return \*this;

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;}

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

data[i][j] = m.data[i][j];

return \*this;

}

template<class T>

inline bool TMatrix<T>::operator==(const TMatrix<T>& m)

{

if (this->len != m.len || len1 != m.len1) return false;

for (int i = 0; i < len1; i++)

{

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

if (data[i][j] != m.data[i][j]) return false;

}

return true;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator+(const TMatrix& m)

{

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;

}

TMatrix res(\*this);

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

res.data[i][j] = this->data[i][j] + m.data[i][j];

return res;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator-(const TMatrix& m)

{

if (this->len != m.len || len1 != m.len1) {

cout << "sizes don't match" << endl;

}

TMatrix res(\*this);

for (int i = 0; i < len1; i++)

for (int j = 0; j < this->len; j++)

res.data[i][j] = this->data[i][j] - m.data[i][j];

return res;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& p)

{

TMatrix<T> res(p.len, len1, 0);

for (int i = 0; i < len1; i++)

{

for (int j = 0; j < p.len; j++)

{

res[i][j] = 0;

for (int l = 0; l < this->len; l++)

res[i][j] += this->data[i][l] \* p.data[l][j];

}

}

return res;

}

template <class T>

TMatrix<T> TMatrix<T>::operator\*(const T a)

{

TMatrix<T> res(\*this);

for (int i = 0; i < len1; i++) {

res[i] = res[i] \* a;

}

return res;

}

/\*template<class T>

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

{

TVector<T> temp(v);

if ((this->len) != temp.GetLen()) throw "problem";

TMatrix<T> res(1, len1, 0);

for (int i = 0; i < len1; i++)

{

for (int j = 0; j < 1; j++)

{

res[i][j] = 0;

for (int x = 0; x < this->len; x++)

res[i][j] += this->data[i][x] \* temp[x];

}

}

return res;

}\*/