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

Структуры хранения матриц специального вида

Выполнил:

студент ф-та ИТММ гр. 3821Б1ФИ3

Дурандин В.Е.

Проверил:

зав. лабораторией СТиВВ

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[**Введение** 3](#_Toc123079754)

[**1.** **Постановка задачи** 4](#_Toc123079755)

[**Руководство пользователя** 5](#_Toc123079759)

[**2.** **Руководство программиста** 7](#_Toc123079760)

[***2.1.*** ***Описание структуры программы*** 7](#_Toc123079761)

[***2.2.*** ***Описание структур данных*** 7](#_Toc123079762)

[**3.** **Эксперименты** 9](#_Toc123079763)

[**3.** **Заключение** 10](#_Toc123079764)

[**4.** **Литература** 11](#_Toc123079765)

[**5.** **Приложения** 12](#_Toc123079766)

[**6.1. Приложение 1 (test\_main.cpp)** 12](#_Toc123079767)

[**6.2. Приложение 2 (vector.h)** 13](#_Toc123079768)

[**6.3. Приложение 3 (matrix.h)** 19](#_Toc123079770)

# **Введение**

Векторы и матрицы в С++ соответственно могут быть представлены разными способами, например, динамические или статические вектора, заданные для хранения вектора; динамические или статические векторы векторов, заданные для хранения матриц; использование готовых библиотек STL и другие. Над матрицами с векторами в данной лабораторной работе предстоит выполнить различные арифметические операции и др.

1. **Постановка задачи**
2. Написать шаблоны классов матрицы и вектора, которые вынесены в статическую библиотеку.
3. Класс матрицы должен быть наследником класса вектор
4. Продемонстрировать их работу на примере (написать в main пример (sample)) и проверить корректность работы на гугл-тестах.

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);

Дополнительные задачи:

1. Реализовать класс треугольные матрицы
2. Реализовать класс ленточных матриц

**Руководство пользователя**

Для создания вектора реализованы четыре конструктора: по умолчанию, копирования и перемещения соответственно.

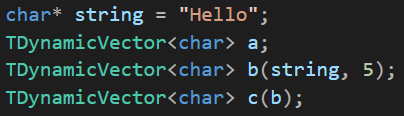


Рисунок 1. Создание объекта класса вектор

Можно узнать длину вектора с помощью метода size():



Рисунок 2. Пример вычисления длины вектора

Есть возможность выполнять разные операции над векторами: индексация, индексация с контролем, сравнение, скалярные операции, векторные операции, ввод/вывод.

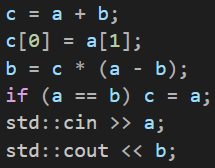


Рисунок 3. Пример векторных операций

Возможно также реализовать различные сортировки в классе вектор.

Для создания матрицы реализован конструктор инициализатор, который создаёт квадратную матрицу.



Рисунок 4. Создание квадратной матрицы заданного размера

Возможны следующие операции над квадратно матрицей: сравнение, матрично-скалярные операции, матрично-векторные операции, матрично-матричные операции, ввод/вывод.

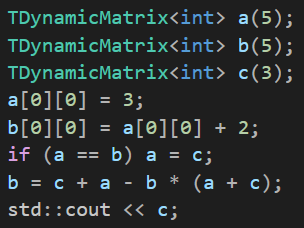


Рисунок 5. Создание квадратных матриц и пример операций над ними

Использование классов нижне-треугольной и ленточных матриц выглядят аналогично, как работа с классом полной матрицы.

1. **Руководство программиста**
   1. ***Описание структуры программы***

Программа состоит из одного решения, в котором есть проекты VectorLib, TestMain, GTestLib.

В проекте VectorLib определено 4 модуля: Lent\_matrix.h; Matrix.h; Triangular\_matrix.h; Vector.h.

* В модуле main.cpp определена стандартная функция int main(), где происходит работа с остальными модулями.
* В модуле Matrix.h определен класс TDynamicMatrix, а также объявлены все его методы и их определения.
* В модуле Vector.h определен класс TDynamicVector, а также объявлены все его методы и их определения.
* В модуле Triangular\_matrix.h определен класс TTriangularDynamicMatrix, а также объявлены все его методы и их определения.
* В модуле Lent\_matrix.h определен класс TLentDynamicMatrix, а также объявлены все его методы и их определения.

В проекте Test определены 5 модулей: test\_main.cpp (запуск гугл-тестов и sample); test\_tLentMatrix.cpp; test\_tmatrix.cpp; test\_tTriangularMatrix.cpp; test\_tvector.cpp. В каждом модуле написаны гугл-тесты для каждого реализованного класса.

В проекте GTestLib определены 2 модуля: вспомогательные статические библиотеки гугл-тестов.

* 1. ***Описание структур данных***

Распишу только класс вектор , так как остальные классы выглядят идентично:

Class TDynamicVector

Внутри класса TDynamicVector определен следующий набор public-методов:

1. Конструктор инициализатор

|  |
| --- |
| TDynamicVector(size\_t size = 1); |

1. Конструктор инициализатор 2

|  |
| --- |
| TDynamicVector(T\* arr, size\_t s); |

1. Конструктор копирования

|  |
| --- |
| TDynamicVector(const TDynamicVector<T>& v); |

1. Конструктор перемещения

|  |
| --- |
| TDynamicVector(TDynamicVector<T>&& v) noexcept; |

1. Деструктор

|  |
| --- |
| ~TDynamicVector(); |

1. Перегрузка операции копирующего равно

|  |
| --- |
| TDynamicVector<T>& operator=(const TDynamicVector<T>& v); |

1. Перегрузка операции перемещающего равно

|  |
| --- |
| TDynamicVector<T>& operator=(TDynamicVector<T>&& v) noexcept; |

1. Метод, возвращающий размер вектора

|  |
| --- |
| size\_t size() const noexcept; |

1. Перегрузка операции индексации

|  |
| --- |
| T& operator[](size\_t ind);  const T& operator[](size\_t ind) const; |

1. Перегрузка операции индексации с контролем

|  |
| --- |
| T& at(size\_t ind);  const T& at(size\_t ind) const; |

1. Перегрузка операций сравнения

|  |
| --- |
| bool operator==(const TDynamicVector<T>& v) const noexcept;  bool operator!=(const TDynamicVector<T>& v) const noexcept; |

1. Перегрузка скалярных операций

|  |
| --- |
| TDynamicVector<T> operator+(T val);  TDynamicVector<T> operator-(double val);  TDynamicVector<T> operator\*(double val); |

1. Перегрузка векторных операций

|  |
| --- |
| TDynamicVector<T> operator+(const TDynamicVector<T>& v);  TDynamicVector<T> operator-(const TDynamicVector<T>& v);  TDynamicVector<T> operator\*(const TDynamicVector<T>& v); |

1. Метод для смены векторов друг с другом

|  |
| --- |
| friend void swap(TDynamicVector<T>& lhs, TDynamicVector<T>& rhs) noexcept; |

1. Перегрузка оператора потокового ввода

|  |
| --- |
| friend std::istream& operator>>(std::istream& istr, TDynamicVector<T>& v); |

1. Перегрузка оператора потокового вывода

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& ostr, const TDynamicVector<T>& v); |

1. **Эксперименты**
2. Оценим время, которое нужно для сложений матриц, асимптотическая сложность которого *O()*:

Таблица 1. Результаты работы сложения матриц.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек). |
| 1000 | 0.0083908 |
| 2000 | 0.0308528 |
| 4000 | 0.11445 |

При увеличении количества элементов, мы видим, что время увеличивается приблизительно в 4 раза.

1. Оценим время, которое нужно для умножения матриц, асимптотическая сложность которого *O()*:

Таблица 2. Результаты работы умножения матриц.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек). |
| 500 | 1.81245 |
| 1000 | 16.868 |
| 2000 | 136.771 |

При увеличении количества элементов, мы видим, что время увеличивается приблизительно в 8 раз.

1. **Заключение**

Удалось реализовать полноценную работу матриц и векторов, а также операции над ними.

Написаны шаблоны классов матрицы и вектора.

Продемонстрирована их работу на примере (в main написан пример).

В программе есть:

* конструкторы (по умолчанию, инициализатор, копирования), деструктор, доступ к защищенным полям;
* перегруженные операции: +, - , \* ,= ,== , [] потоковый ввод и вывод;
* перегруженные операции +, - ,\* реализованы для векторов (вектор +-\* вектор), матриц (матрица +-\* матрица), матрично-векторные (матрица \* вектор и наоборот);

1. **Литература**
2. Объектно-ориентированное программирование – Режим доступа:

<https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование> - Загл. с экрана

1. Язык программирования С++ - Режим доступа:

<https://ru.wikipedia.org/wiki/C%2B%2B> – Загл. с экрана

1. **Приложения**

**6.1. Приложение 1 (test\_main.cpp)**

#include <gtest.h>

#include <iostream>

#include "Triangular\_matrix.h"

#include "Lent\_matrix.h"

#include "Vector.h"

#include "Matrix.h"

using namespace std;

int main(int argc, char \*\*argv)

{

TLentDynamicMatrix<int> g(6, 3);

g[0][0] = 4; g[0][1] = 8; g[0][2] = 9; g[0][3] = 6;

g[1][0] = 7; g[1][1] = 8; g[1][2] = 6; g[1][3] = 5; g[1][4] = 4;

g[2][0] = 1; g[2][1] = 2; g[2][2] = 3; g[2][3] = 4; g[2][4] = 5; g[2][5] = 6;

g[3][0] = 7; g[3][1] = 6; g[3][2] = 8; g[3][3] = 3; g[3][4] = 2;

g[4][0] = 3; g[4][1] = 7; g[4][2] = 2; g[4][3] = 1;

int n = 3;

TDynamicVector<int> a(n);

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

{

a[i] = i + 1;

}

TDynamicVector<int> b = (a \* 2);

TDynamicVector<int> c;

c = a + b;

std::cout << "Vector a + b = \n";

for (auto i = c.begin(); i != c.end(); ++i)

{

std::cout << \*i << ' ';

}

std::cout << std::endl;

TDynamicMatrix<int> m(n);

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

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

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

std::cout << "Matrix m =: \n" << m;

TDynamicMatrix<int> x(m), res(n);

std::cout <<"\nMatrix x =: \n" << x;

res = x \* m;

std::cout << "\nMatrix x \* m =: \n" << res;

//std::cout << res;

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**6.2. Приложение 2 (vector.h)**

#pragma once

#include <iostream>

const int MAX\_VECTOR\_SIZE = 100000000;

template<typename T>

class TVectorIterator;

// Динамический вектор -

// шаблонный вектор на динамической памяти

template<typename T>

class TDynamicVector

{

protected:

size\_t sz;

T\* pMem;

public:

TDynamicVector(size\_t size = 1);

TDynamicVector(T\* arr, size\_t s);

TDynamicVector(const TDynamicVector<T>& v);

TDynamicVector(TDynamicVector<T>&& v) noexcept;

~TDynamicVector();

TDynamicVector<T>& operator=(const TDynamicVector<T>& v);

TDynamicVector<T>& operator=(TDynamicVector<T>&& v) noexcept;

size\_t size() const noexcept;

// индексация

T& operator[](size\_t ind);

const T& operator[](size\_t ind) const;

// индексация с контролем

T& at(size\_t ind);

const T& at(size\_t ind) const;

// сравнение

bool operator==(const TDynamicVector<T>& v) const noexcept;

bool operator!=(const TDynamicVector<T>& v) const noexcept;

// скалярные операции

TDynamicVector<T> operator+(T val);

TDynamicVector<T> operator-(double val);

TDynamicVector<T> operator\*(double val);

// векторные операции

TDynamicVector<T> operator+(const TDynamicVector<T>& v);

TDynamicVector<T> operator-(const TDynamicVector<T>& v);

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

friend void swap(TDynamicVector<T>& lhs, TDynamicVector<T>& rhs) noexcept;

// ввод/вывод

template<typename T>

friend std::istream& operator>>(std::istream& istr, TDynamicVector<T>& v);

template<typename T>

friend std::ostream& operator<<(std::ostream& ostr, const TDynamicVector<T>& v);

TVectorIterator<T> begin();

TVectorIterator<T> end();

};

template<typename T>

inline TDynamicVector<T>::TDynamicVector(size\_t size)

{

if (size <= 0 || size > MAX\_VECTOR\_SIZE) throw "TDynamicVector(size\_t size) size <= 0 || size > MAX\_VECTOR\_SIZE";

sz = size;

pMem = new T[sz];

}

template<typename T>

inline TDynamicVector<T>::TDynamicVector(T\* arr, size\_t s)

{

if (size <= 0) throw "TDynamicVector(T\* arr, size\_t s) size <= 0";

if (arr == nullptr) throw "TDynamicVector(T\* arr, size\_t s) arr == 0";

sz = s;

pMem = new T[sz];

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

{

pMem[i] = arr[i];

}

}

template<typename T>

inline TDynamicVector<T>::TDynamicVector(const TDynamicVector<T>& v)

{

if (v.pMem == nullptr)

{

sz = 0;

pMem = nullptr;

}

else

{

sz = v.sz;

pMem = new T[sz];

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

{

pMem[i] = v.pMem[i];

}

}

}

template<typename T>

inline TDynamicVector<T>::TDynamicVector(TDynamicVector<T>&& v) noexcept

{

pMem = v.pMem;

sz = v.sz;

v.sz = 0;

v.pMem = nullptr;

}

template<typename T>

inline TDynamicVector<T>::~TDynamicVector()

{

if (pMem != nullptr) delete[] pMem;

sz = 0;

}

template<typename T>

inline TDynamicVector<T>& TDynamicVector<T>::operator=(const TDynamicVector<T>& v)

{

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

if (pMem != nullptr) delete[] pMem;

if (v.pMem == nullptr)

{

sz = 0;

pMem = nullptr;

}

else

{

sz = v.sz;

pMem = new T[sz];

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

{

pMem[i] = v.pMem[i];

}

}

return \*this;

}

template<typename T>

inline TDynamicVector<T>& TDynamicVector<T>::operator=(TDynamicVector<T>&& v) noexcept

{

if (pMem != nullptr) delete[] pMem;

pMem = v.pMem;

sz = v.sz;

v.pMem = nullptr;

v.sz = 0;

return \*this;

}

template<typename T>

inline size\_t TDynamicVector<T>::size() const noexcept

{

return sz;

}

template<typename T>

inline T& TDynamicVector<T>::operator[](size\_t ind)

{

return pMem[ind];

}

template<typename T>

inline const T& TDynamicVector<T>::operator[](size\_t ind) const

{

return pMem[ind];

}

template<typename T>

inline T& TDynamicVector<T>::at(size\_t ind)

{

if (pMem == nullptr) throw "opertator[] pMem == nullptr";

if (ind < 0 || ind >= sz) throw "opertator[] out of range";

return pMem[ind];

}

template<typename T>

inline const T& TDynamicVector<T>::at(size\_t ind) const

{

if (pMem == nullptr) throw "opertator[] pMem == nullptr";

if (ind < 0 || ind >= sz) throw "opertator[] out of range";

return pMem[ind];

}

template<typename T>

inline bool TDynamicVector<T>::operator==(const TDynamicVector<T>& v) const noexcept

{

if (sz != v.sz) return false;

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

{

if (pMem[i] != v.pMem[i]) return false;

}

return true;

}

template<typename T>

inline bool TDynamicVector<T>::operator!=(const TDynamicVector<T>& v) const noexcept

{

return !(this->operator==(v));

}

template<typename T>

inline TDynamicVector<T> TDynamicVector<T>::operator+(T val)

{

if (pMem == nullptr) throw "operator+(T val) pMem == nullptr";

TDynamicVector tmp(\*this);

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

{

tmp.pMem[i] += val;

}

return tmp;

}

template<typename T>

inline TDynamicVector<T> TDynamicVector<T>::operator-(double val)

{

if (pMem == nullptr) throw "operator-(double val) pMem == nullptr";

TDynamicVector<T> tmp(\*this);

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

{

tmp.pMem[i] -= val;

}

return tmp;

}

template<typename T>

inline TDynamicVector<T> TDynamicVector<T>::operator\*(double val)

{

if (pMem == nullptr) throw "operator\*(double val) pMem == nullptr";

TDynamicVector<T> tmp(\*this);

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

{

tmp.pMem[i] \*= val;

}

return tmp;

}

template<typename T>

inline TDynamicVector<T> TDynamicVector<T>::operator+(const TDynamicVector<T>& v)

{

if (pMem == nullptr) throw "operator+(const TDynamicVector& v) pMem == nullptr";

if (sz != v.sz) throw "operator+(const TDynamicVector& v) sz != v.sz";

TDynamicVector<T> tmp(\*this);

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

{

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

}

return tmp;

}

template<typename T>

inline TDynamicVector<T> TDynamicVector<T>::operator-(const TDynamicVector<T>& v)

{

if (pMem == nullptr) throw "operator-(const TDynamicVector& v) pMem == nullptr";

if (sz != v.sz) throw "operator-(const TDynamicVector& v) sz != v.sz";

TDynamicVector<T> tmp(\*this);

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

{

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

}

return tmp;

}

template<typename T>

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

{

if (pMem == nullptr) throw "operator\*(const TDynamicVector& v) pMem == nullptr";

if (sz != v.sz) throw "operator\*(const TDynamicVector& v) sz != v.sz";

T res = 0;

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

{

res += pMem[i] \* v.pMem[i];

}

return res;

}

template<typename T>

inline TVectorIterator<T> TDynamicVector<T>::begin()

{

return TVectorIterator<T>(\*this);

}

template<typename T>

inline TVectorIterator<T> TDynamicVector<T>::end()

{

return TVectorIterator<T>(\*this, this->size());

}

template<typename T>

void swap(TDynamicVector<T>& lhs, TDynamicVector<T>& rhs) noexcept

{

std::swap(lhs.sz, rhs.sz);

std::swap(lhs.pMem, rhs.pMem);

}

template<typename T>

std::istream& operator>>(std::istream& istr, TDynamicVector<T>& v)

{

for (size\_t i = 0; i < v.sz; i++)

istr >> v.pMem[i]; // требуется оператор>> для типа T

return istr;

}

template<typename T>

std::ostream& operator<<(std::ostream& ostr, const TDynamicVector<T>& v)

{

for (size\_t i = 0; i < v.sz; i++)

ostr << v.pMem[i] << ' '; // требуется оператор<< для типа T

return ostr;

}

//==============================================================================================================//

template<typename T>

class TVectorIterator

{

public:

TVectorIterator(TDynamicVector<T>& v);

TVectorIterator(TDynamicVector<T>& v, int ind);

TVectorIterator(TVectorIterator<T>& p);

bool operator == (const TVectorIterator<T>& v);

bool operator != (const TVectorIterator<T>& v);

TVectorIterator<T> operator++ ();

TVectorIterator<T> operator-- ();

T& operator\* ();

private:

TDynamicVector<T>& vector;

int index;

};

template<typename T>

inline TVectorIterator<T>::TVectorIterator(TDynamicVector<T>& v) : vector(v)

{

index = 0;

}

template<typename T>

inline TVectorIterator<T>::TVectorIterator(TDynamicVector<T>& v, int ind) : vector(v)

{

index = ind;

}

template<typename T>

inline TVectorIterator<T>::TVectorIterator(TVectorIterator<T>& p) : vector(p.vector), index(p.index)

{

}

template<typename T>

inline bool TVectorIterator<T>::operator==(const TVectorIterator<T>& v)

{

if (&vector == &(v.vector) && index == v.index) return true;

return false;

}

template<typename T>

inline bool TVectorIterator<T>::operator!=(const TVectorIterator<T>& v)

{

return !(this->operator==(v));

}

template<typename T>

inline TVectorIterator<T> TVectorIterator<T>::operator++()

{

this->index++;

if (this->index > vector.size()) this->index = vector.size();

return \*this;

}

template<typename T>

inline TVectorIterator<T> TVectorIterator<T>::operator--()

{

this->index--;

if (this->index < 0) this->index = 0;

return \*this;

}

template<typename T>

inline T& TVectorIterator<T>::operator\*()

{

return vector.at(index);

}

**6.3. Приложение 3 (matrix.h)**

#pragma once

#include "Vector.h"

const int MAX\_MATRIX\_SIZE = 10000;

// Динамическая матрица -

// шаблонная матрица на динамической памяти

template<typename T>

class TDynamicMatrix : private TDynamicVector<TDynamicVector<T>>

{

using TDynamicVector<TDynamicVector<T>>::pMem;

using TDynamicVector<TDynamicVector<T>>::sz;

public:

TDynamicMatrix(size\_t s = 1);

using TDynamicVector<TDynamicVector<T>>::operator[];

using TDynamicVector<TDynamicVector<T>>::size;

using TDynamicVector<TDynamicVector<T>>::at;

// сравнение

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

// матрично-скалярные операции

TDynamicMatrix<T> operator\*(const T& val);

// матрично-векторные операции

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

// матрично-матричные операции

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

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

TDynamicMatrix<T> operator\*(const TDynamicMatrix<T>& m);

// ввод/вывод

template<typename T>

friend std::istream& operator>>(std::istream& istr, TDynamicMatrix<T>& v);

template<typename T>

friend std::ostream& operator<<(std::ostream& ostr, const TDynamicMatrix<T>& v);

};

template<typename T>

inline TDynamicMatrix<T>::TDynamicMatrix(size\_t s) : TDynamicVector<TDynamicVector<T>>(s)

{

if (s > MAX\_MATRIX\_SIZE) throw "s > MAX\_MATRIX\_SIZE";

for (size\_t i = 0; i < sz; i++)

pMem[i] = TDynamicVector<T>(sz);

}

template<typename T>

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

{

return TDynamicVector<TDynamicVector<T>>::operator == (m);

}

template<typename T>

inline TDynamicMatrix<T> TDynamicMatrix<T>::operator\*(const T& val)

{

TDynamicMatrix(sz) res;

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

{

res.pMem[i] = pMem[i] \* val;

}

return res;

}

template<typename T>

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

{

TDynamicVector<T> res(sz);

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

{

res.pMem[i] = pMem[i] \* v;

}

return res;

}

template<typename T>

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

{

if (this->sz != m.sz) throw "operator+(const TDynamicMatrix<T>& m) sz != m.sz";

TDynamicMatrix<T> res(sz);

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

{

res.pMem[i] = pMem[i] + m.pMem[i];

}

return res;

}

template<typename T>

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

{

if (this->sz != m.sz) throw "operator-(const TDynamicMatrix<T>& m) sz != m.sz";

TDynamicMatrix<T> res(sz);

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

{

res.pMem[i] = pMem[i] - m.pMem[i];

}

return res;

}

template<typename T>

inline TDynamicMatrix<T> TDynamicMatrix<T>::operator\*(const TDynamicMatrix<T>& m)

{

if (this->sz != m.sz) throw "operator\*(const TDynamicMatrix<T>& m) sz != m.sz";

TDynamicMatrix<T> res(sz);

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

{

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

{

res[i][j] = 0;

for (int k = 0; k < this->sz; k++)

{

res[i][j] += this->pMem[i][k] \* m.pMem[k][j];

}

}

}

return res;

}

template<typename T>

inline std::istream& operator>>(std::istream& istr, TDynamicMatrix<T>& v)

{

std::cout << "Enter " << v.sz \* v.sz << " values of Matrix: \n";

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

for (int j = 0; j < v.sz; j++)

istr >> v.pMem[i][j];

return istr;

}

template<typename T>

inline std::ostream& operator<<(std::ostream& ostr, const TDynamicMatrix<T>& v)

{

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

{

for (int j = 0; j < v.sz; j++)

ostr << v.pMem[i][j] << '\t';

std::cout << std::endl;

}

return ostr;

}