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

Балясов И.А.

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\_tmatrix.cpp)** 12](#_Toc123079767)

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

[**6.3. Приложение 3 (TriangleMatrix.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

**Руководство пользователя**

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

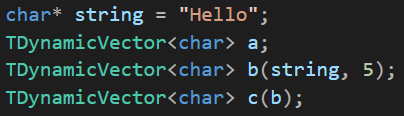


Рисунок 1. Создание объекта класса вектор

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.

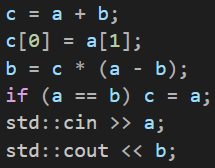


Рисунок 3. Пример векторных операций

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.

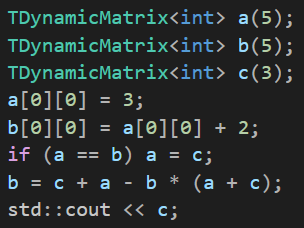


Рисунок 5. Создание квадратных матриц и пример операций над ними

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 Vector, VectorLib, Test, GTestLib.

В проекте Vector определен 1 модуль: main.cpp определена стандартная функция int main(), где происходит работа с остальными модулями.

В проекте VectorLib определено 3 модуля: 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 проекте Test определены 3 модуля: test\_main.cpp (запуск гугл-тестов и sample); test\_tmatrix.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 | 0.0083238 |
| 2000 | 0.0307525 |
| 4000 | 0.13475 |

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 | 1.81344 |
| 1000 | 15.654 |
| 2000 | 134.876 |

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\_tmatrix.cpp)**

#include "Matrix.h"

#include <gtest.h>

TEST(TDynamicMatrix, can\_create\_matrix\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TDynamicMatrix<int> m(5));

}

TEST(TDynamicMatrix, cant\_create\_too\_large\_matrix)

{

ASSERT\_ANY\_THROW(TDynamicMatrix<int> m(MAX\_MATRIX\_SIZE + 1));

}

TEST(TDynamicMatrix, throws\_when\_create\_matrix\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TDynamicMatrix<int> m(-5));

}

TEST(TDynamicMatrix, can\_create\_copied\_matrix)

{

TDynamicMatrix<int> m(5);

ASSERT\_NO\_THROW(TDynamicMatrix<int> m1(m));

}

TEST(TDynamicMatrix, copied\_matrix\_is\_equal\_to\_source\_one)

{

TDynamicMatrix<int>m1(2);

for(int i = 0; i < 2; i++)

for(int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(m1);

for(int i = 0; i < 2; i++)

for(int j = 0; j < 2; j++)

EXPECT\_EQ(m1[i][j] , m2[i][j]);

}

TEST(TDynamicMatrix, copied\_matrix\_has\_its\_own\_memory)

{

TDynamicMatrix<int>m1(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(m1);

m2[0][0] = -1;

EXPECT\_NE(m1[0][0],m2[0][0]);

}

TEST(TDynamicMatrix, can\_get\_size)

{

TDynamicMatrix<int>m(2);

EXPECT\_EQ(2, m.size());

}

TEST(TDynamicMatrix, can\_set\_and\_get\_element)

{

TDynamicMatrix<int>m(2);

m[1][1] = 5;

EXPECT\_EQ(5, m[1][1]);

}

TEST(TDynamicMatrix, throws\_when\_set\_element\_with\_negative\_index)

{

TDynamicMatrix<int>m(2);

ASSERT\_ANY\_THROW(m.at(1).at(-1));

ASSERT\_ANY\_THROW(m.at(-1).at(1));

}

TEST(TDynamicMatrix, throws\_when\_set\_element\_with\_too\_large\_index)

{

TDynamicMatrix<int>m(2);

ASSERT\_ANY\_THROW(m.at(1).at(2));

ASSERT\_ANY\_THROW(m.at(2).at(1));

}

TEST(TDynamicMatrix, can\_assign\_matrix\_to\_itself)

{

TDynamicMatrix<int>m1(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(m1);

m2 = m2;

for(int i = 0;i < 2; i++)

for(int j = 0; j < 2; j++)

EXPECT\_EQ(m1[i][j], m2[i][j]);

}

TEST(TDynamicMatrix, can\_assign\_matrices\_of\_equal\_size)

{

TDynamicMatrix<int>m1(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(2);

m2 = m1;

for(int i = 0; i < 2; i++)

for(int j = 0; j < 2; j++)

EXPECT\_EQ(m1[i][j], m2[i][j]);

}

TEST(TDynamicMatrix, assign\_operator\_change\_matrix\_size)

{

TDynamicMatrix<int>m1(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(5);

int size = m2.size();

m2 = m1;

int new\_size = m2.size();

EXPECT\_NE(size, new\_size);

}

TEST(TDynamicMatrix, can\_assign\_matrices\_of\_different\_size)

{

TDynamicMatrix<int>m1(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

m1[i][j] = i + j + 1;

TDynamicMatrix<int>m2(5);

m2 = m1;

EXPECT\_EQ(2, m2.size());

for(int i = 0; i < m1.size(); i++)

for(int j = 0;j<m1.size();j++)

EXPECT\_EQ(m1[i][j], m2[i][j]);

}

TEST(TDynamicMatrix, compare\_equal\_matrices\_return\_true)

{

TDynamicMatrix<int>m1(2);

TDynamicMatrix<int>m2(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

{

m1[i][j] = i + j + 1;

m2[i][j] = i + j + 1;

}

EXPECT\_EQ(true , m1==m2);

}

TEST(TDynamicMatrix, compare\_matrix\_with\_itself\_return\_true)

{

TDynamicMatrix<int>m(2);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

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

EXPECT\_EQ(true , m==m);

}

TEST(TDynamicMatrix, matrices\_with\_different\_size\_are\_not\_equal)

{

TDynamicMatrix<int>m(2);

TDynamicMatrix<int>m1(3);

EXPECT\_EQ(false, m==m1);

}

TEST(TDynamicMatrix, can\_add\_matrices\_with\_equal\_size)

{

TDynamicMatrix<int>m1(2);

TDynamicMatrix<int>m2(2);

for(int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

{

m1[i][j] = i + j + 1;

m2[i][j] = i + j + 2;

}

TDynamicMatrix<int>m3 = m1 + m2;

for(int i = 0; i < 2; i++)

for(int j = 0;j<2;j++)

EXPECT\_EQ(m1[i][j] + m2[i][j], m3[i][j]);

}

TEST(TDynamicMatrix, cant\_add\_matrices\_with\_not\_equal\_size)

{

TDynamicMatrix<int>m1(2);

TDynamicMatrix<int>m2(3);

ASSERT\_ANY\_THROW(TDynamicMatrix<int>m3 = m1 + m2);

}

TEST(TDynamicMatrix, can\_subtract\_matrices\_with\_equal\_size)

{

TDynamicMatrix<int>m1(2);

TDynamicMatrix<int>m2(2);

for(int i = 0; i < 2; i++)

for(int j = 0; j < 2; j++)

{

m1[i][j] = i + j + 1;

m2[i][j] = i + j + 2;

}

TDynamicMatrix<int>m3 = m2 - m1;

for(int i = 0; i < 2; i++)

for(int j = 0; j < 2; j++)

EXPECT\_EQ(m2[i][j] - m1[i][j], m3[i][j]);

}

TEST(TDynamicMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size)

{

TDynamicMatrix<int>m(2);

TDynamicMatrix<int>m1(3);

ASSERT\_ANY\_THROW(TDynamicMatrix<int>m2 = m - m1);

}

**6.2. Приложение 2 (vector.h)**

#pragma once

#include "Vector.h"

template<typename T>

class TTrMatrix : public TDynamicVector<TDynamicVector<T>>

{

public:

TTrMatrix(size\_t size = 1);

virtual ~TTrMatrix();

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

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

template<typename T>

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

template<typename T>

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

};

template<typename T>

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

{

if (size < 1)

throw "size < 1";

pMem = new TDynamicVector<T>[size];

sz = size;

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

pMem[i] = TDynamicVector<T>(i + 1);

}

template<typename T>

inline TTrMatrix<T>::~TTrMatrix()

{

}

template<typename T>

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

{

if (sz != p.sz)

throw "operator\* : sz != p.sz";

TTrMatrix<T> res(sz);

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

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

if (i >= j)

{

res[i][j] = 0;

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

if ((k <= i) && (k >= j))

res[i][j] += (\*this)[i][k] \* p[k][j];

}

return res;

}

template<typename T>

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

{

if (sz != p.sz)

throw "operator+ : sz != p.sz";

TTrMatrix<T> res(sz);

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

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

if (i >= j)

res[i][j] = (\*this)[i][j] + p[i][j];

return res;

}

template<typename T>

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

{

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

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

if (i >= j)

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

return istr;

}

template<typename T>

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

{

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

{

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

if (i >= j)

ostr << v.pMem[i][j] << ' ';

ostr << std::endl;

}

return ostr;

}

**6.3. Приложение 3 (TriangleMatrix.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;

}