МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
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**

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе №2

***Структуры хранения матриц специального вида***

**Выполнил:** студент группы

3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. Е. Филатьев

Подпись

**Проверила:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Я.В. Силенко

Подпись

Нижний Новгород  
2023

**Содержание**

[1.Введение: 3](#_Toc149903450)

[2. Постановка задачи: 4](#_Toc149903451)

[3.Описание алгоритмов: 5](#_Toc149903452)

[3.1. Операции над векторами: 5](#_Toc149903453)

[3.2. Операции над матрицами: 5](#_Toc149903454)

[4.Описание программы: 6](#_Toc149903455)

[4.1. Класс вектор: 6](#_Toc149903456)

[4.2. Класс матрица: 7](#_Toc149903457)

[4.3. Класс таймер: 7](#_Toc149903458)

[5.Результаты: 8](#_Toc149903459)

[6.Заключение 9](#_Toc149903460)

[7.Литература: 10](#_Toc149903461)

[8.Приложение: 11](#_Toc149903462)

1.Введение:

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), который 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 его элементы. Количество строк и столбцов задает размер матрицы. Матрицу можно также представить в виде функции двух дискретных аргументов. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Матрицы широко применяются в математике для компактной записи систем линейных [алгебраических](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5) или [дифференциальных](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение [систем линейных уравнений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9) сводится к операциям над матрицами.

2. Постановка задачи:

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц и выполнение основных операций над ними:

* Сложение/вычитание;
* Умножение;
* Копирование;
* Сравнение.

Програмные средства должны содержать:

* Класс вектор (на шаблонах);
* Класс матрица (на шаблонах);
* Тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними.

3.Описание алгоритмов:

3.1. Операции над векторами:

Пусть заданы два вектора A = (a1, a2, ..., an) и B = (b1, b2, ..., bn). Рассмотрим следующие основные операции над векторами:

* Сравнение (A = B). Вектора считаются равными тогда и только тогда, когда ai = bi при всех i = 1..n.
* Прибавление скаляра (A + a). Результатом сложения вектора A и скаляра, а называется вектор A’ = (a1 + a, a2 + a, ..., an + a).
* Вычитание скаляра (A – a). Результатом вычитания вектора A и скаляра, а называется вектор A’ = (a1 – a, a2 – a, ..., an – a).
* Умножение на скаляр (A \* a). Результатом умножения вектора A на скаляр, а называется вектор A’ = (a1 \* a, a2 \* a, ..., an \* a).
* Вычисление длины. Длиной вектора A называется скалярная величина

* Сложение векторов (A + B). Результатом сложения векторов A и B называется вектор С = (a1 + b1, a2 + b2, ..., an + bn).
* Вычитание векторов (A – B). Результатом вычитания векторов A и B называется вектор С = (a1 – b1, a2 – b2, ..., an – bn).
* Скалярное произведение векторов (A \* B). Скалярным произведением векторов A и B называется скалярная величина

3.2. Операции над матрицами:

Пусть заданы две матрицы A = (aij) и B = (bij) и вектор V = (vj), где i = 1..m; j = 1..n. Рассмотрим следующие основные операции над матрицами:

* Сравнение (A = B). Матрицы считаются равными тогда и только тогда, когда aij = bij при всех i = 1..m; j = 1..n.
* Умножение на скаляр (A \* d). Результатом умножения матрицы A на скаляр d называется матрица D = (dij), где dij = aij \* d, при всех i = 1..m; j = 1..n.
* Умножение на вектор (A \* V). Результатом умножения матрицы A на вектор V называется вектор D = (di), где , при всех i = 1..m; j = 1..n.
* Сложение матриц (A + B). Результатом сложения матриц A и B называется матрица C = (cij), где cij = aij + bij при всех i = 1..m; j = 1..n.
* Вычитание матриц (A – B). Результатом вычитания матриц A и B называется матрица C = (cij), где cij = aij – bij при всех i = 1..m; j = 1..n.
* Умножение матриц (A \* B). Результатом умножения матриц A = (aij) и B = (bjk) называется матрица C = (cik), где cij = aij \* bjk при всех i = 1..m; j = 1..n; k = 1..n.

4.Описание программы:

Программа состоит из двух основных классов и нескольких побочных:

4.1. Класс вектор:

Класс вектор – шаблонный класс с изменяемым типом хранения содержаший:

Поля:

* Size\_t sz – размер вектора
* T\* pMem – указатель шаблонный тип для выделения памяти для массива

Методы:

* TDynamicVector(size\_t size = 1) : sz(size) – Конструктор по умолчанию
* TDynamicVector(T\* arr, size\_t s) : sz(s) – Конструктор инициализации
* TDynamicVector(const TDynamicVector& v) – Конструктор копирования
* TDynamicVector(TDynamicVector&& v) noexcept – Конструктор перемещения
* TDynamicVector& operator=(const TDynamicVector& v) – Оператор присваивания
* TDynamicVector& operator=(TDynamicVector&& v) noexcept – Перемещающий оператор присваивания
* size\_t size() const noexcept – Метод возвращающий размер вектора
* T& operator[](size\_t ind) – Оператор индексирования без зашиты
* const T& operator[](size\_t ind) const – Константный оператор индексации без защиты
* T& at(size\_t ind) Оператор индексирования с зашитой
* const T& at(size\_t ind) const Константный оператор индексации с защиты
* bool operator==(const TDynamicVector& v) const noexcept - Оператор сравнения векторов
* bool operator!=(const TDynamicVector& v) const noexcept- Оператор сравнения векторов
* TDynamicVector operator+(T val) – Скалярный оператор сложения вектора с константой
* TDynamicVector operator-(T val) – Скалярный оператор вычитания вектора с константой
* TDynamicVector operator\*(T val) – Скалярный оператор умножения вектора с константой
* TDynamicVector operator+(const TDynamicVector& v) – Векторный оператор сложения двух векторов
* TDynamicVector operator-(const TDynamicVector& v) – Векторный оператор вычитания двух векторов
* T operator\*(const TDynamicVector& v) noexcept(noexcept(T())) – Векторный оператор умножения двух векторов
* friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept – Дружественная функция обменом значения между двумя векторами
* friend istream& operator>>(istream& istr, TDynamicVector& v) – Дружественная функция ввода
* friend ostream& operator<<(ostream& ostr, const TDynamicVector& v) – Дружественная функция вывода

4.2. Класс матрица:

* TDynamicMatrix(size\_t s = 1) : TDynamicVector<TDynamicVector<T>>(s) – Конструктор инициализации
* bool operator==(const TDynamicMatrix& m) const noexcept – Оператор сравнения двух матриц
* bool operator!=(const TDynamicMatrix& m) const – Оператор сравнения двух матриц
* TDynamicMatrix operator\*(const T& val) – Скалярный оператор умножения матрицы на скаляр
* TDynamicVector<T> operator\*(const TDynamicVector<T>& v) – Векторное операция умножения матрицы на вектор
* TDynamicMatrix operator+(const TDynamicMatrix& m) – Матричная операция сложения двух матриц
* TDynamicMatrix operator-(const TDynamicMatrix& m) – Матричная операция разности двух матриц
* TDynamicMatrix operator\*(const TDynamicMatrix& m) – Матричная операция умножения двух матриц
* friend istream& operator>>(istream& istr, TDynamicMatrix& v) – Дружественная функция ввода матрицы
* friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v) – Дружественная функция вывода матрицы
* size\_t size() const noexcept – Метод получения размера матрицы
* T& at(size\_t ind1, size\_t ind2) – Оператор индексирования с зашитой
* const T& at(size\_t ind) const – Константный оператор индексирования с зашитой

4.3. Класс таймер:

* Timer() – Конструктор инициализации таймера
* void set() – Метод получения времени работы программы

5.Результаты:

Передадим в функцию, которая умножает матрицы сгенерированный случайным образом и проверим результат с предположительной скоростью. Предположительная скорость умножения двух матриц O(n3).

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время (Т(n)) | О(n) | T(n) / O(n) |
| 10 | 2.28 \* 10-5 | 1000 | 2.28 \*10-8 |
| 100 | 4.934\*10-3 | 106 | 4.934\*10-9 |
| 300 | 0.1553 | 27 \* 106 | 5.75\*10-9 |
| 500 | 0.855 | 125\*106 | 6.84\*10-9 |
| 700 | 1.99 | 346\*106 | 5.75\*10-9 |
| 1000 | 9.473 | 109 | 9.47\*10-9 |
| 2000 | 105.417 | 8\*109 | 13.1\*10-9 |

Построим график:

Получаем прямую, значит данная сложность совпадает с предположенной скоростью.

6.Заключение

Мы реализовали класс вектор и класс матриц, их основные методы и действия над ним. Реализовали тесты для проверки работоспособности данной программы. Замерили время выполнения программ для различных размеров матриц и проверили с предположенной скоростью выполнения умножения матриц. Проверили работоспособность программы и проверили обработку ошибки при неверном вводе.

7.Литература:

Википедия - <https://ru.wikipedia.org/wiki/Вектор_(геометрия)>

Википедия - <https://ru.wikipedia.org/wiki/Матрица_(математика)>

Разница между throw и assert - <https://www.programmersought.com/article/54774475923/>

Конструктор перемещения - <https://metanit.com/cpp/tutorial/14.2.php>

Обработка ошибок и исключений - <https://learn.microsoft.com/ru-ru/cpp/cpp/errors-and-exception-handling-modern-cpp?view=msvc-170>

8.Приложение:

Класс вектор:

template<typename T>

class TDynamicVector

{

protected:

size\_t sz;

T\* pMem;

public:

TDynamicVector(size\_t size = 1) : sz(size)

{

pMem = nullptr;

if (size == 0)

throw out\_of\_range("Vector size should be greater than zero");

if (size > MAX\_VECTOR\_SIZE)

throw length\_error("The size of the vector must be less than MAX\_VECTOR\_SIZE");

pMem = new T[size]();// {}; // У типа T д.б. констуктор по умолчанию

}

TDynamicVector(T\* arr, size\_t s) : sz(s){

pMem = nullptr;

if (sz == 0)

throw out\_of\_range("Vector size should be greater than zero");

if (sz > MAX\_VECTOR\_SIZE)

throw length\_error("The size of the vector must be less than MAX\_VECTOR\_SIZE")

assert(arr != nullptr && "TDynamicVector ctor requires non-nullptr arg");

pMem = new T[sz];

std::copy(arr, arr + sz, pMem);

}

TDynamicVector(const TDynamicVector& v)

{

this->sz = v.sz;

if (v.pMem == nullptr)

this->pMem = nullptr;

else {

this->pMem = new T[sz];

std::copy(v.pMem, v.pMem + sz, pMem);

}

}

TDynamicVector(TDynamicVector&& v) noexcept

{

this->pMem = nullptr;

swap(\*this, v);

}

~TDynamicVector()

{

if (pMem != nullptr)

delete[]pMem;

sz = 0;

pMem = nullptr;

}

TDynamicVector& operator=(const TDynamicVector& v)

{

if (this == &v)

return \*this;

this->sz = v.sz;

if (v.pMem == nullptr)

this->pMem = nullptr;

else {

this->pMem = new T[sz];

std::copy(v.pMem, v.pMem + sz, pMem);

}

return \*this;

}

TDynamicVector& operator=(TDynamicVector&& v) noexcept

{

swap(\*this, v);

return \*this;

}

size\_t size() const noexcept { return sz; }

// индексация

T& operator[](size\_t ind)

{

return pMem[ind];

}

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

{

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if (ind < 0 || ind >= sz)

throw out\_of\_range("Vector size should be greater than zero");

return pMem[ind];

}

const T& at(size\_t ind) const

{

if (ind < 0 || ind >= sz)

throw out\_of\_range("Vector size should be greater than zero");

return pMem[ind];

}

// сравнение

bool operator==(const TDynamicVector& v) const noexcept

{

if (this->sz != v.sz)

return false;

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

if (this->pMem[i] != v.pMem[i])

return false;

return true;

}

bool operator!=(const TDynamicVector& v) const noexcept

{

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

}

// скалярные операции

TDynamicVector operator+(T val)

{

TDynamicVector temp(this->sz);

if (pMem != nullptr) {

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

temp.pMem[i] = this->pMem[i] + val;

}

}

return temp;

}

TDynamicVector operator-(T val)

{

TDynamicVector temp(this->sz);

if (pMem != nullptr) {

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

temp.pMem[i] = this->pMem[i] - val;

}

}

return temp;

}

TDynamicVector operator\*(T val)

{

TDynamicVector temp(this->sz);

if (pMem != nullptr) {

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

temp.pMem[i] = this->pMem[i] \* val;

}

}

return temp;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

if (this->sz != v.sz)

throw exception("Вектора различного размера");

TDynamicVector temp(this->sz);

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

temp.pMem[i] = this->pMem[i] + v.pMem[i];

return temp;

}

TDynamicVector operator-(const TDynamicVector& v)

{

if (this->sz != v.sz)

throw exception("Вектора различного размера");

TDynamicVector temp(this->sz);

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

temp.pMem[i] = this->pMem[i] - v.pMem[i];

return temp;

}

T operator\*(const TDynamicVector& v) noexcept(noexcept(T()))

{

if (this->sz != v.sz)

throw exception("Вектора различного размера");

T temp{};

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

temp += pMem[i] \* v[i];

return temp;

}

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept

{

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

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

}

// ввод/вывод

friend istream& operator>>(istream& istr, TDynamicVector& v)

{

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

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

return istr;

}

friend ostream& operator<<(ostream& ostr, const TDynamicVector& v)

{

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

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

return ostr;

}

};

Класс матрица:

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) : TDynamicVector<TDynamicVector<T>>(s)

{

if (sz == 0)

throw out\_of\_range("Matrix size should be greater than zero");

if (sz > MAX\_MATRIX\_SIZE)

throw length\_error("The size of the matrix must be less than MAX\_MATRIX\_SIZE");

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

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

}

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

// сравнение

bool operator==(const TDynamicMatrix& m) const noexcept

{

if (this->sz != m.sz)

return false;

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

if (pMem[i] != m.pMem[i])

return false;

}

return true;

}

bool operator!=(const TDynamicMatrix& m) const noexcept {

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

}

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix temp(\*this);

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

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

return temp;

}

// матрично-векторные операции

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

{

if (this->sz != v.size())

throw exception("Различные линейные размеры вектора и матрицы");

TDynamicVector<T> temp(sz);

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

temp[i] = this->pMem[i] \* v;

}

return temp;

}

// матрично-матричные операции

TDynamicMatrix operator+(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw exception("Матрицы различного размера");

TDynamicMatrix temp(sz);

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

temp[i] = pMem[i] + m[i];

}

return temp;

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw exception("Матрицы различного размера");

TDynamicMatrix temp(sz);

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

temp[i] = pMem[i] - m[i];

}

return temp;

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw exception("Матрицы различного размера");

TDynamicMatrix temp(sz);

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

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

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

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

}

}

return temp;

}

// ввод/вывод

friend istream& operator>>(istream& istr, TDynamicMatrix& v)

{

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

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

istr >> v[i][j];

}

}

return istr;

}

friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v)

{

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

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

ostr << v[i][j] << " ";

}

ostr << endl;

}

return ostr;

}

size\_t size() const noexcept { return sz; }

//Индексация

T& at(size\_t ind1, size\_t ind2)

{

if (ind1 < 0 || ind1 >= sz || ind2 < 0 || ind2 >= sz)

throw out\_of\_range("Vector size should be greater than zero");

return pMem[ind1][ind2];

}

const T& at(size\_t ind) const

{

if (ind1 < 0 || ind1 >= sz || ind2 < 0 || ind2 >= sz)

throw out\_of\_range("Vector size should be greater than zero");

return pMem[ind1][ind2];

}

Класс таймер:

class Timer {

private:

chrono::time\_point<chrono::steady\_clock> start, end;

public:

Timer() {

start = chrono::high\_resolution\_clock::now();

}

void set() {

end = chrono::high\_resolution\_clock::now();

chrono::duration<float> dur = end - start;

cout << "Время: " << dur.count() << " secund" << endl;

}

};