МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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**

**Отчет по лабораторной работе**

на тему:

**«Шаблонная матрица на динамической памяти»**

**Выполнил:** студент группы 3822Б1ПР2

Наумов Богдан Александрович

Нижний Новгород

2023

Оглавление

[1. Введение 3](#_Toc153277379)

[2. Постановка задачи 4](#_Toc153277380)

[3. Руководство пользователя 5](#_Toc153277381)

[4. Руководство программиста 7](#_Toc153277382)

[4.1.Описание структуры программы 7](#_Toc153277383)

[4.2.Описание структур данных 7](#_Toc153277384)

[4.3.Описание алгоритмов 9](#_Toc153277385)

[5. Эксперименты 10](#_Toc153277386)

[6. Заключение 11](#_Toc153277387)

[7. Приложение 12](#_Toc153277388)

## Введение

Матрицы - один из фундаментальных инструментов линейной алгебры и науки о данных. Матрицы представляют собой удобный способ организации и работы с числовыми данными и могут использоваться для решения линейных преобразований, одновременных линейных уравнений и множества других задач, относящихся ко многим областям математики и прикладной науки. В данной статье рассматриваются варианты реализации матричных классов на основе шаблонных динамических векторов.

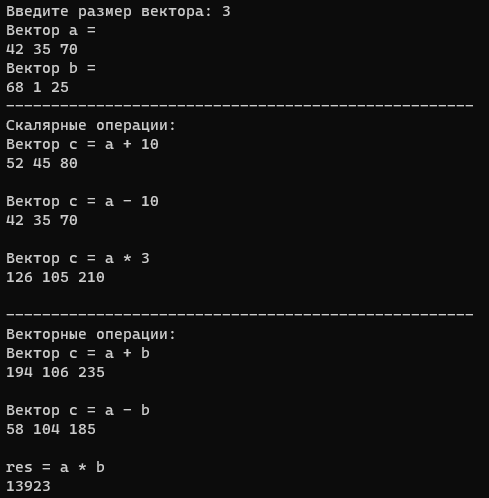
## Постановка задачи

Чтобы выполнить работу нужно решить следующие задачи:

1. Реализовать вспомогательный шаблонный класс TDynamicVector.
2. Реализовать шаблонный класс TDynamicMatrix.
3. Написать пример программы, которая будет демонстрировать работу класса TDynamicMatrix.
4. Написать набор тестов с использованием GoogleC++ TestingFramework для проверки работы классов TDynamicVector и TDynamicMatrix.

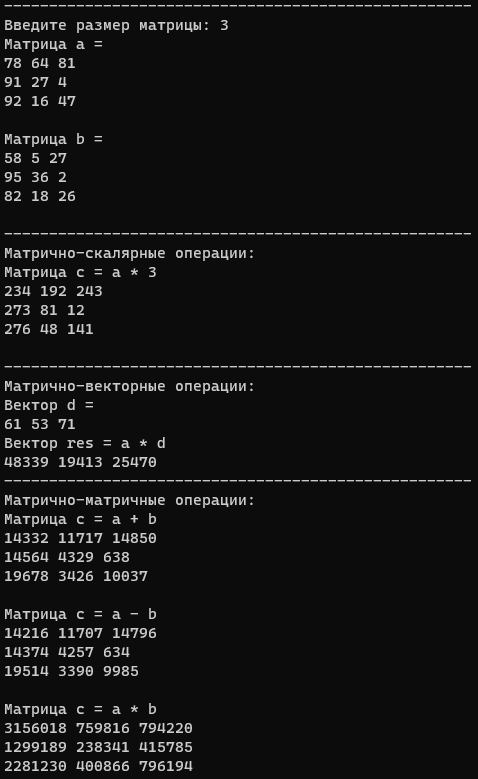
## Руководство пользователя

При запуске программы, демонстрирующей работу классов TDynamicVector и TDynamicMatrix, она просит ввести размер вектора. После чего она создает два вектора указанного размера, заполняет их случайными числами и выполняет операции над ними.



*Рис.1 (вывод случайно заданных векторов и результатов операций над ними)*

Далее программа запрашивает размер матрицы, создает две матрицы указанного размера, заполняет их случайными числами и выполняет операции над ними.



*Рис.2 (вывод случайно заданных матриц и результатов операций над ними)*

## 4. Руководство программиста

## 4.1.Описание структуры программы

Программа состоит из следующих модулей:

* Модуль **include**. Статическая библиотека. Включает в себя заголовочный файл tmatrix.h, в котором описаны методы с реализацией классов TDynamicVector и TDynamicMatrix.
* Модули **test\_vector** и **test\_matrix**. Набор тестов для реализованных классов. Включает в себя файлы **VectorTest.cpp** и **MatrixTest.cpp**. Реализованы они с помощью использования фреймворка **Google Test**.
* Модуль **samples** - пример использования классов. Включает в себя файл с реализацией **sample\_matrix.cpp**.
  1. **Описание структуры программы**

Программа состоит из следующих модулей:

1. Модуль **Include**. Статическая библиотека. Включает в себя заголовочный файл **tmatrix.h**, в котором описаны методы с реализацией шаблонного класса вектор **TDynamicVector** и **TDynamicMatrix**.
2. Модули **test\_vector** и **test\_matrix**. Набор тестов для обоих классов. Включает в себя файлы **VectorTest.cpp** и **MatrixTest.cpp**. Реализованы они с помощью использования фреймворка **Google Test**.
3. Модуль **samples** - пример использования классов. Включает в себя файл с реализацией **sample\_matrix.cpp**.
   1. **Описание структур данных**

Класс **tmatrix.h**:

Поля:

size\_t sz; - размер вектора/матрицы.

T\* pMem; – динамический массив для хранения вектора/матрицы.

Конструкторы для вектора:

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

{

if (sz < 0)

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

if (sz > MAX\_VECTOR\_SIZE) {

throw "cant\_create\_too\_large\_vector";

}

pMem = new T[sz]();

}

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

{

pMem = new T[sz];

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

}

Конструктор копирования для вектора:

TDynamicVector(const TDynamicVector& v)

{

if (v.pMem == nullptr) {

sz = 0;

pMem = nullptr;

}

else {

sz = v.sz;

pMem = new T[sz];

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

}

}

Конструктор перемещения для вектора:

TDynamicVector(TDynamicVector&& v) noexcept

{

sz = v.sz;

pMem = v.pMem;

v.sz = 0;

v.pMem = nullptr;

}

Конструктор для матрицы:

TDynamicMatrix(size\_t s = 1) : TDynamicVector<TDynamicVector<T>>(s)

{

if (sz < 0)

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

if (sz > MAX\_MATRIX\_SIZE) {

throw "cant\_create\_too\_large\_matrix";

}

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

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

}

Деструктор для вектора:

~TDynamicVector()

{

if (pMem != nullptr) {

delete[] pMem;

sz = 0;

}

}

Методы для вектора:

size\_t size();

Перегрузки для вектора:

T& operator[](int ind); - перегрузка оператора []

bool operator==(const TDynamicVector& v); - перегрузка проверки на равенство 2ух векторов

bool operator!=(const TDynamicVector& v); - перегрузка проверки на неравенство 2ух векторов

TDynamicVector operator+(T val); - перегрузка оператора сложения

TDynamicVector operator-(T val); - перегрузка оператора вычитания числа из вектора

TDynamicVector operator\*(T val); - перегрузка оператора сложения числа с вектором

TDynamicVector operator+(const TDynamicVector& v); - перегрузка оператора сложения векторов

TDynamicVector operator-(const TDynamicVector& v); - перегрузка оператора вычитания векторов

T operator\*(const TDynamicVector& v); - перегрузка оператора умножения векторов

friend istream& operator>>(istream& istr, TDynamicVector& v); - ввод

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

Методы для матрицы:

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

Перегрузки для матрицы:

bool operator==(const TDynamicMatrix& m); - перегрузка проверки на равенство 2ух векторов

TDynamicMatrix operator\*(const T& val); - перегрузка сложения числа с матрицей

TDynamicVector<T> operator\*(const TDynamicVector<T>& v); - перегрузка умножения вектора с матрицей

TDynamicMatrix operator+(const TDynamicMatrix& m); - перегрузка сложения матриц

TDynamicMatrix operator-(const TDynamicMatrix& m); - перегрузка вычитания матриц

TDynamicMatrix<T> operator\*(const TDynamicMatrix<T> & m); - перегрузка умножения матриц

friend istream& operator>>(istream& istr, TDynamicMatrix& v); - ввод матрицы

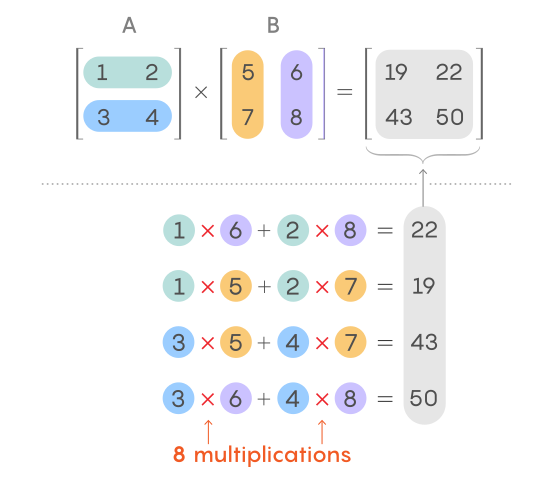
friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v); - вывод матрицы

## 4.3.Описание алгоритмов

Умножение матриц:

В результате произведения матриц A и B получается матрица C = AB, такая, что C[i][j] (элементы j-го столбца i-й строки) равна сумме произведений элементов i-й строки матрицы A и соответствующих элементов j-го столбца матрицы B.

Перегрузка оператора умножения на матрицу осуществляется с помощью трех вложенных циклов: первый цикл проходит по строкам левой матрицы, второй цикл проходит по столбцам правой матрицы, а третий цикл проходит по элементам текущего столбца правой матрицы.



## 5. Эксперименты

Эксперименты проводились на ПК со следующими параметрами:

1. Операционная система: Windows 11
2. AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
3. Версия Visual Studio: 2022

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Время работы оператора сложения (в млс)  O(n2) | Время работы оператора умножения (в млс)  O(n3) |
| 1000 | 5 | 1431 |
| 3000 | 41 | 42516 |
| 6000 | 169 | 361594 |

Мы использовали классические методы для сложения и умножения матриц. Их сложности соответственно O(n2) и O(n3). Из результатов в таблице видно, что цифры, полученные в ходе экспериментов, соответствуют теоретическим значениям.

## 6. Заключение

В этой лаборатории все задачи были решены. Два класса - шаблонные векторы и матрицы; класс матриц основан на векторах. Самостоятельная работа на практике позволила мне углубиться в аппарат автоматизированного тестирования. Это позволило мне проверить правильность работы реализованных мною классов. Также была оценена производительность метода хранения матриц.

## 7. Приложение

// ННГУ, ИИТММ, Курс "Алгоритмы и структуры данных"

//

// Copyright (c) Сысоев А.В.

//

//

#ifndef \_\_TDynamicMatrix\_H\_\_

#define \_\_TDynamicMatrix\_H\_\_

#include <iostream>

**using** **namespace** std;

**const** **int** MAX\_VECTOR\_SIZE = **100000000**;

**const** **int** MAX\_MATRIX\_SIZE = **10000**;

// Динамический вектор -

// шаблонный вектор на динамической памяти

**template**<**typename** T>

**class** **TDynamicVector**

{

**protected:**

**size\_t** sz;

T\* pMem;

**public:**

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

{

**if** (sz == **0**)

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

**if** (sz > MAX\_VECTOR\_SIZE)

**throw** length\_error("Exceeded vector max size");

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

}

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

{

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

**if** (sz > MAX\_VECTOR\_SIZE)

**throw** length\_error("Exceeded vector max size");

pMem = **new** T[sz];

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

}

TDynamicVector(**const** TDynamicVector& v)

{

**if** (v.pMem == nullptr) {

sz = **0**;

pMem = nullptr;

}

**else** {

sz = v.sz;

pMem = **new** T[sz];

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

}

}

TDynamicVector(TDynamicVector&& v) noexcept

{

sz = v.sz;

pMem = nullptr;

std::swap(pMem, v.pMem);

}

~TDynamicVector()

{

**if** (pMem != nullptr)

**delete**[] pMem;

pMem = nullptr;

sz = **0**;

}

TDynamicVector& **operator**=(**const** TDynamicVector& v)

{

**if** (**this** == &v)

**return** \***this**;

**if** (sz != v.sz)

{

**if** (pMem != nullptr)

**delete**[] pMem;

sz = v.sz;

pMem = **new** T[sz];

}

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

**return** \***this**;

}

TDynamicVector& **operator**=(TDynamicVector&& v) noexcept

{

**if** (**this** == &v)

**return** \***this**;

**if** (pMem != nullptr) {

**delete**[] pMem;

pMem = nullptr;

}

sz = v.sz;

std::swap(pMem, v.pMem);

**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 > sz || ind < **0**) **throw** "Index not in range";

**return** pMem[ind];

}

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

{

**if** (ind > sz || ind < **0**) **throw** "Index not in range";

**return** pMem[ind];

}

// сравнение

**bool** **operator**==(**const** TDynamicVector& 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;

}

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

{

**return** !(\***this** == v);

}

// скалярные операции

TDynamicVector **operator**+(T val)

{

TDynamicVector result(sz);

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

result.pMem[i] = pMem[i] + val;

}

**return** result;

}

TDynamicVector **operator**-(T val)

{

TDynamicVector result(sz);

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

result.pMem[i] = pMem[i] - val;

}

**return** result;

}

TDynamicVector **operator**\*(T val)

{

TDynamicVector result(sz);

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

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

}

**return** result;

}

// векторные операции

TDynamicVector **operator**+(**const** TDynamicVector& v)

{

**if** (sz != v.sz) **throw** "Error len vector";

TDynamicVector **result**(sz);

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

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

}

**return** result;

}

TDynamicVector **operator**-(**const** TDynamicVector& v)

{

**if** (sz != v.sz) **throw** "Error len vector";

TDynamicVector **result**(sz);

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

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

}

**return** result;

}

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

{

**if** (sz != v.sz) **throw** "Error len vector";

T result = T();

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

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

}

**return** result;

}

**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];

**return** istr;

}

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

{

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

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

**return** ostr;

}

};

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

{

**if** (sz > MAX\_MATRIX\_SIZE)

**throw** length\_error("Exceeded matrix max size");

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

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

}

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

{

**if** (sz > MAX\_MATRIX\_SIZE)

**throw** length\_error("Exceeded matrix max size");

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

pMem = **new** T[sz];

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

}

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

// сравнение

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

{

**if** (sz != m.sz) **return** false;

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

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

}

**return** true;

}

// матрично-скалярные операции

TDynamicMatrix **operator**\*(**const** T& val)

{

TDynamicMatrix result(sz);

**for** (**int** i = **0**; i < size(**this**); i++) {

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

}

**return** result;

}

TDynamicMatrix **operator**/(**const** T& val)

{

TDynamicMatrix<**double**>result(sz);

**for** (**int** i = **0**; i < size(**this**); i++)

{

result.pMem[i] = pMem[i] / val;

}

**return** result;

}

// матрично-векторные операции

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

{

**if** (sz != v.sz) **throw** "Error len";

TDynamicVector<T> result(sz);

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

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

result[i] += pMem[i][j] \* v[j];

}

}

**return** result;

}

// матрично-матричные операции

TDynamicMatrix **operator**+(**const** TDynamicMatrix& m)

{

**if** (sz != m.sz) **throw** "Error len";

TDynamicMatrix **result**(sz);

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

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

}

**return** result;

}

TDynamicMatrix **operator**-(**const** TDynamicMatrix& m)

{

**if** (sz != m.sz) **throw** "Error len";

TDynamicMatrix **result**(sz);

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

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

}

**return** result;

}

TDynamicMatrix **operator**\*(**const** TDynamicMatrix& m)

{

**if** (sz != m.sz) **throw** "Error len";

TDynamicMatrix **result**(sz);

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

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

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

result[i][j] += pMem[i][k] \* m.pMem[k][j];

}

}

}

**return** result;

}

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

// ввод/вывод

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

{

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

istr >> v.pMem[i];

}

**return** istr;

}

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

{

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

ostr << v.pMem[i] << endl;

}

**return** ostr;

}

};

#endif

sample\_matrix.cpp

// ННГУ, ИИТММ, Курс "Алгоритмы и структуры данных"

//

// Copyright (c) Сысоев А.В.

//

// Тестирование матриц

#include <iostream>

#include "tmatrix.h"

//---------------------------------------------------------------------------

**void** **main**()

{

**int** i, j, size;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование класс работы с матрицами" << endl;

cout << "Введите размер вектора: ";

cin >> size;

TDynamicVector<**int**> Va(size);

TDynamicVector<**int**> Vb(size);

TDynamicVector<**int**> Vc(size);

**for** (i = **0**; i < size; i++) {

Va[i] = rand() % **100** + **1**;

Vb[i] = rand() % **100** + **1**;

}

cout << "Вектор а = " << endl << Va << endl;

cout << "Вектор b = " << endl << Vb << endl;

cout << "----------------------------------------------------" << endl;

cout << "Скалярные операции:" << endl;

Vc = Va + **10**;

cout << "Вектор c = а + 10" << endl << Vc << endl << endl;

Vc = Va - **10**;

cout << "Вектор c = а - 10" << endl << Vc << endl << endl;

Vc = Va \* **3**;

cout << "Вектор c = а \* 3" << endl << Vc << endl << endl;

cout << "----------------------------------------------------" << endl;

cout << "Векторные операции:" << endl;

Vc = Va + Vb;

cout << "Вектор c = а + b" << endl << Vc << endl << endl;

Vc = Va - Vb;

cout << "Вектор c = а - b" << endl << Vc << endl << endl;

**int** res = Va \* Vb;

cout << "res = а \* b" << endl << res << endl << endl;

cout << "----------------------------------------------------" << endl;

cout << "Введите размер матрицы: ";

cin >> size;

TDynamicVector<**int**> Vres(size);

TDynamicVector<**int**> Vd(size);

TDynamicMatrix<**int**> Ma(size);

TDynamicMatrix<**int**> Mb(size);

TDynamicMatrix<**int**> Mc(size);

**for** (i = **0**; i < size; i++)

**for** (j = **0**; j < size; j++)

{

Ma[i][j] = rand() % **100**;

Mb[i][j] = rand() % **100**;

Vd[i] = rand() % **100**;

}

**if** (size <= **10**) cout << "Матрица а = " << endl << Ma << endl;

**if** (size <= **10**) cout << "Матрица b = " << endl << Mb << endl;

cout << "----------------------------------------------------" << endl;

cout << "Матрично-скалярные операции:" << endl;

Mc = Ma \* **3**;

**if** (size <= **10**) cout << "Матрица c = а \* 3" << endl << Mc << endl;

cout << "----------------------------------------------------" << endl;

cout << "Матрично-векторные операции:" << endl;

**if** (size <= **10**) cout << "Вектор d = " << endl << Vd << endl;

Vres = Ma \* Vd;

**if** (size <= **10**) cout << "Вектор res = а \* d" << endl << Vres << endl;

cout << "----------------------------------------------------" << endl;

cout << "Матрично-матричные операции:" << endl;

Mc = Ma + Mb;

**if** (size <= **10**) cout << "Матрица c = а + b" << endl << Mc << endl;

Mc = Ma - Mb;

**if** (size <= **10**) cout << "Матрица c = а - b" << endl << Mc << endl;

**clock\_t** start1 = clock();

Mc = Ma \* Mb;

**clock\_t** stop1 = clock();

// double seconds = (double)(stop1 - start1) / CLOCKS\_PER\_SEC;

// cout << "Времени ушло на сложение:" << endl << seconds << endl;

**if** (size <= **10**) cout << "Матрица c = а \* b" << endl << Mc << endl;

}

//---------------------------------------------------------------------------