ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Сибирский государственный университет телекоммуникаций и информатики (СибГУТИ)

Расчетно-графическая работа

по дисциплине: «Программирование»

Выполнил:

студент 1 курса

группы ИКС-433

Нгуен Зуй Ань Куеевич

Преподаватель:

Вейлер Андрей Игоревич

Новосибирск, 2025

Требования к РГР 1.

1. Все программы реализуются на языке С.
2. На оценку «3»: решить поставленную задачу согласно варианту, оформить отчёт.
3. На оценку «4»: в дополнение к предыдущей оценке добавить проверку всех входных данных на корректность. В случае ошибки должно выдаваться сообщение, поясняющее ее суть для пользователя. По возможности создать многофайловый проект и скомпилировать статическую/динамическую библиотеку из функций, выполняющих поставленную задачу.
4. На оценку «5»: в дополнение к предыдущей оценке сборка проекта осуществляется с использованием CMake. Покрыть UNIT-тестами код

Задание:

Создать программу, которая вычисляет обратную матрицу. Программа принимает в качестве аргумента командной строки имя файла, содержащего матрицу в удобном для восприятия виде. По завершении работы программа должна вывести обратную матрицу или сообщить о невозможности ее вычисления.

Критерии оценки:

Оценка «хорошо»:

- Выполнена предварительная проверка размера матрицы.

- Реализована проверка результирующей матрицы путем получения единичной матрицы при перемножении результата с исходной матрицей.

- Обязательно динамическое выделение памяти для входных данных.

Оценка «отлично»: В дополнение к критериям оценки «хорошо» проект собран с использованием CMake.

Анализ задачи:

Структура состоит из:

matrix.h - заголовочный файл с объявлениями функций и структур.

matrix.c - реализация функций для работы с матрицами.

main.c - основная программа, которая читает матрицу из файла и вычисляет обратную матрицу.

test_matrix.c - unit-тесты для проверки функций из matrix.c.

CMakeLists.txt - конфигурация для сборки основного проекта.

test/CMakeLists.txt - конфигурация для сборки тестов.

matrix.txt - пример входного файла с матрицей.

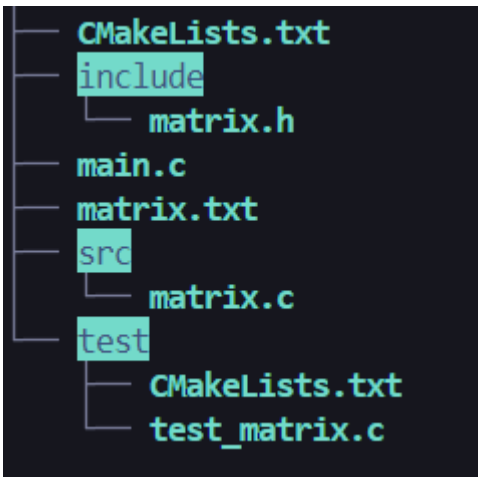


Рис1. Дерево файлов

Содержимое каждого файла:

```c
#ifndef MATRIX_H
#define MATRIX_H
#include <math.h>

// Структура для хранения матрицы
typedef struct {
    double** data;
    int rows;
    int cols;
} Matrix;

// Функции для работы с матрицами
Matrix* create_matrix(int rows, int cols);
void free_matrix(Matrix* matrix);
Matrix* read_matrix_from_file(const char* filename);
void print_matrix(const Matrix* matrix);
int is_matrix_square(const Matrix* matrix);
Matrix* create_identity_matrix(int size);
int inverse_matrix(Matrix* matrix, Matrix* inverse);
Matrix* copy_matrix(const Matrix* matrix);
int multiply_matrices(const Matrix* a, const Matrix* b, Matrix* result);
int are_matrices_equal(const Matrix* a, const Matrix* b, double epsilon);

#endif
```

Рис2. файл matrix.h

```
find_library(CMOCKA_LIBRARY cmocka)
find_path(CMOCKA_INCLUDE_DIR cmocka.h)

if(CMOCKA_LIBRARY AND CMOCKA_INCLUDE_DIR)
    message(STATUS "Found cmocka: ${CMOCKA_LIBRARY}")

    # Создаем тестовый исполняемый файл
    add_executable(test_matrix
        test_matrix.c
        ../src/matrix.c
    )

    # Подключаем заголовочные файлы
    target_include_directories(test_matrix PRIVATE ../include ${CMOCKA_INCLUDE_DIR})

    # Подключаем cmocka
    target_link_libraries(test_matrix ${CMOCKA_LIBRARY})

    # Добавляем тест
    add_test(NAME matrix_tests COMMAND test_matrix)
else()
    message(WARNING "cmocka не найден, тестирование нельзя сделать")
endif()
```

Рис3. файл src/CmakeLists.txt

```
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include "matrix.h"
#include <math.h>

// Тест создания и освобождения матрицы
static void test_create_and_free_matrix(void** state) {
    Matrix* matrix = create_matrix(3, 3);
    assert_non_null(matrix);
    assert_int_equal(matrix->rows, 3);
    assert_int_equal(matrix->cols, 3);

    free_matrix(matrix);
}

// Тест проверки квадратной матрицы
static void test_is_matrix_square(void** state) {
    Matrix* square = create_matrix(2, 2);
    assert_non_null(square);
    assert_true(is_matrix_square(square));

    Matrix* non_square = create_matrix(2, 3);
    assert_non_null(non_square);
    assert_false(is_matrix_square(non_square));
```

Рис4. Файл src/test_matrix.c

```
cmake_minimum_required(VERSION 3.10)
project(inverse_matrix C)

set(CMAKE_C_STANDARD 11)
set(CMAKE_C_STANDARD_REQUIRED ON)

# Основная программа
add_executable(inverse_matrix
    src/matrix.c
    main.c
)

# Подключаем заголовочные файлы
target_include_directories(inverse_matrix PRIVATE include)

enable_testing()

add_subdirectory(test)
```

Рис5. Файл CMakeLists.txt

```
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Использование: %s <имя_файла_с_матрицей>\n", argv[0]);
        return 1;
    }

    Matrix* matrix = read_matrix_from_file(argv[1]);
    if (matrix == NULL) {
        return 1;
    }

    // Проверка, что матрица квадратная
    if (!is_matrix_square(matrix)) {
        printf("Ошибка: матрица должна быть квадратной\n");
        free_matrix(matrix);
        return 1;
    }

    // Создание матрицы для обратной матрицы
    Matrix* inverse = create_matrix(matrix->rows, matrix->cols);
    if (inverse == NULL) {
        printf("Ошибка: не удалось создать матрицу для обратной\n");
        free_matrix(matrix);
```

Рис6. Файл main.c

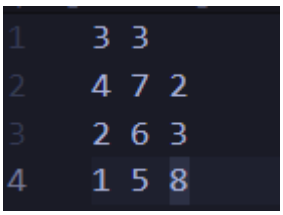


Рис7. Файл matrix.txt

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "matrix.h"

// Создание матрицы
Matrix* create_matrix(int rows, int cols) {
    if (rows <= 0 || cols <= 0) {
        return NULL;
    }

    Matrix* matrix = (Matrix*)malloc(sizeof(Matrix));
    if (matrix == NULL) {
        return NULL;
    }

    matrix->rows = rows;
    matrix->cols = cols;

    matrix->data = (double**)malloc(rows * sizeof(double*));
    if (matrix->data == NULL) {
        free(matrix);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
```

Рис8. Файл matrix.c

Основные функции

Функции в matrix.c:

create_matrix - создает матрицу заданного размера.

```c
Matrix* create_matrix(int rows, int cols) {
    if (rows <= 0 || cols <= 0) {
        return NULL;
    }

    Matrix* matrix = (Matrix*)malloc(sizeof(Matrix));
    if (matrix == NULL) {
        return NULL;
    }

    matrix->rows = rows;
    matrix->cols = cols;

    matrix->data = (double**)malloc(rows * sizeof(double*));
    if (matrix->data == NULL) {
        free(matrix);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
        matrix->data[i] = (double*)malloc(cols * sizeof(double));
        if (matrix->data[i] == NULL) {
            for (int j = 0; j < i; j++) {
                free(matrix->data[j]);
            }
            free(matrix->data);
            free(matrix);
            return NULL;
        }
    }

    return matrix;
```

Рис9. Функция **create_matrix**

free_matrix - освобождает память, выделенную под матрицу.

```
void free_matrix(Matrix* matrix) {
    if (matrix == NULL) {
        return;
    }

    for (int i = 0; i < matrix->rows; i++) {
        free(matrix->data[i]);
    }
    free(matrix->data);
    free(matrix);
```

Рис10. Функция **free_matrix**

read_matrix_from_file - читает матрицу из файла.

```
Matrix* read_matrix_from_file(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Ошибка: не удалось открыть файл %s\n", filename);
        return NULL;
    }

    int rows, cols;
    if (fscanf(file, "%d %d", &rows, &cols) != 2) {
        printf("Ошибка: неверный формат файла\n");
        fclose(file);
        return NULL;
    }

    Matrix* matrix = create_matrix(rows, cols);
    if (matrix == NULL) {
        printf("Ошибка: не удалось создать матрицу\n");
        fclose(file);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (fscanf(file, "%lf", &matrix->data[i][j]) != 1) {
                printf("Ошибка: неверные данные в файле\n");
                free_matrix(matrix);
                fclose(file);
                return NULL;
```

Рис11. функция **read_matrix_from_file**

print_matrix - выводит матрицу на экран.

```
void print_matrix(const Matrix* matrix) {
    if (matrix == NULL) {
        printf("Матрица не существует\n");
        return;
    }

    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            printf("%8.4f ", matrix->data[i][j]);
        }
        printf("\n");
    }
}
```

Рис12. Функция **print_matrix**

is_matrix_square - проверяет, является ли матрица квадратной.

```
int is_matrix_square(const Matrix* matrix) {
    if (matrix == NULL) {
        return 0;
    }
    return matrix->rows == matrix->cols;
}
```

Рис13. Функция **is_matrix_square**

create_identity_matrix - создает единичную матрицу.

```
Matrix* create_identity_matrix(int size) {
    Matrix* matrix = create_matrix(size, size);
    if (matrix == NULL) {
        return NULL;
    }

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix->data[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }

    return matrix;
}
```

Рис14. Функция **create_identity_matrix**

copy_matrix - создает копию матрицы.

```
Matrix* copy_matrix(const Matrix* matrix) {
    if (matrix == NULL) {
        return NULL;
    }

    Matrix* copy = create_matrix(matrix->rows, matrix->cols);
    if (copy == NULL) {
        return NULL;
    }

    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            copy->data[i][j] = matrix->data[i][j];
        }
    }

    return copy;
}
```

Рис15. Функция **copy_matrix**

inverse_matrix - вычисляет обратную матрицу методом Гаусса.

```
int inverse_matrix(Matrix* matrix, Matrix* inverse) {
    if (matrix == NULL || inverse == NULL || matrix->rows != matrix->cols
        inverse->rows != inverse->cols || matrix->rows != inverse->rows) {
        return 0;
    }

    int n = matrix->rows;
    double temp;

    // Присоединяем единичную матрицу справа
    Matrix* augmented = create_matrix(n, 2 * n);
    if (augmented == NULL) {
        return 0;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            augmented->data[i][j] = matrix->data[i][j];
            augmented->data[i][j + n] = (i == j) ? 1.0 : 0.0;
        }
    }

    //  метод Гаусса
    for (int i = 0; i < n; i++) {
        // Если элемент на диагонали равен 0, ищем строку для замены
        if (fabs(augmented->data[i][i]) < 1e-10) {
            int swap_row = -1;
```

Рис16. Функция **inverse_matrix**

multiply_matrices - умножает две матрицы.

```c
int multiply_matrices(const Matrix* a, const Matrix* b, Matrix* result) {
    if (a == NULL || b == NULL || result == NULL ||
        a->cols != b->rows || result->rows != a->rows || result->cols != b->cols) {
        return 0;
    }

    for (int i = 0; i < a->rows; i++) {
        for (int j = 0; j < b->cols; j++) {
            result->data[i][j] = 0.0;
            for (int k = 0; k < a->cols; k++) {
                result->data[i][j] += a->data[i][k] * b->data[k][j];
            }
        }
    }

    return 1;
}
```

Рис17. Функция **multiply_matrices**

are_matrices_equal - проверяет равенство матриц с заданной точностью.

```c
int are_matrices_equal(const Matrix* a, const Matrix* b, double epsilon) {
    if (a == NULL || b == NULL || a->rows != b->rows || a->cols != b->cols) {
        return 0;
    }

    for (int i = 0; i < a->rows; i++) {
        for (int j = 0; j < a->cols; j++) {
            if (fabs(a->data[i][j] - b->data[i][j]) > epsilon) {
                return 0;
            }
        }
    }

    return 1;
}
```

Рис18. Функция **are_matrices_equal**

Команды для сборки и запуска

Проект собирается с помощью CMake.

mkdir build && cd build

cmake .. - Конфигурирует проект

cmake .. -DBUILD_TESTS=ON – Конфигурирует проект с тестами

make - Собирает проект

./inverse_matrix ../matrix.txt - Запускает программы

make test - Запускает тест(**cmocka)**

Тестирование

Тесты в test_matrix.c проверяют:

```c
static void test_create_and_free_matrix(void** state) {
    Matrix* matrix = create_matrix(3, 3);
    assert_non_null(matrix);
    assert_int_equal(matrix->rows, 3);
    assert_int_equal(matrix->cols, 3);

    free_matrix(matrix);
}
```

Рис19. Создание и освобождение матрицы.

```c
static void test_is_matrix_square(void** state) {
    Matrix* square = create_matrix(2, 2);
    assert_non_null(square);
    assert_true(is_matrix_square(square));

    Matrix* non_square = create_matrix(2, 3);
    assert_non_null(non_square);
    assert_false(is_matrix_square(non_square));

    free_matrix(square);
    free_matrix(non_square);
}
```

Рис20. Проверку на квадратность.

```c
static void test_create_identity_matrix(void** state) {
    Matrix* identity = create_identity_matrix(3);
    assert_non_null(identity);

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (i == j) {
                assert_true(fabs(identity->data[i][j] - 1.0) < 1e-10);
            } else {
                assert_true(fabs(identity->data[i][j]) < 1e-10);
            }
        }
    }

    free_matrix(identity);
}
```

Рис21. Создание единичной матрицы.

```c
static void test_copy_matrix(void** state) {
    Matrix* original = create_matrix(2, 2);
    assert_non_null(original);

    original->data[0][0] = 1.0;
    original->data[0][1] = 2.0;
    original->data[1][0] = 3.0;
    original->data[1][1] = 4.0;

    Matrix* copy = copy_matrix(original);
    assert_non_null(copy);

    assert_true(are_matrices_equal(original, copy, 1e-10));

    free_matrix(original);
    free_matrix(copy);
}
```

Рис22 Копирование матрицы.

```c
static void test_inverse_matrix(void** state) {
    Matrix* matrix = create_matrix(2, 2);
    assert_non_null(matrix);

    matrix->data[0][0] = 4.0;
    matrix->data[0][1] = 7.0;
    matrix->data[1][0] = 2.0;
    matrix->data[1][1] = 6.0;

    Matrix* inverse = create_matrix(2, 2);
    assert_non_null(inverse);

    assert_true(inverse_matrix(matrix, inverse));

    // Ожидаемая обратная матрица
    Matrix* expected_inverse = create_matrix(2, 2);
    assert_non_null(expected_inverse);

    expected_inverse->data[0][0] = 0.6;
    expected_inverse->data[0][1] = -0.7;
    expected_inverse->data[1][0] = -0.2;
    expected_inverse->data[1][1] = 0.4;

    assert_true(are_matrices_equal(inverse, expected_inverse, 1e-6));

    // Проверка, что произведение матрицы на обратную дает единичную матри
    Matrix* identity = create_matrix(2, 2);
    assert_non_null(identity);
```

Рис23. Корректность вычисления обратной матрицы.

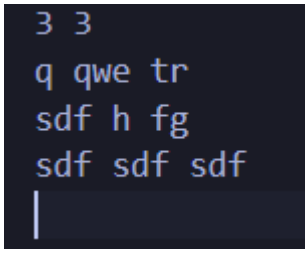
```c
static void test_singular_matrix(void** state) {
    Matrix* matrix = create_matrix(2, 2);
    assert_non_null(matrix);

    matrix->data[0][0] = 1.0;
    matrix->data[0][1] = 2.0;
    matrix->data[1][0] = 2.0;
    matrix->data[1][1] = 4.0;

    Matrix* inverse = create_matrix(2, 2);
    assert_non_null(inverse);

    assert_false(inverse_matrix(matrix, inverse));

    free_matrix(matrix);
    free_matrix(inverse);
}
```

Рис24. Обработку вырожденной матрицы (не должна иметь обратной).

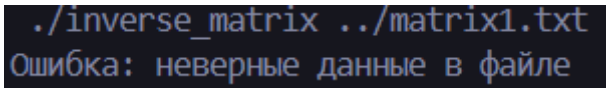
Тесты используют библиотеку **cmocka**.

Обработка ошибок:

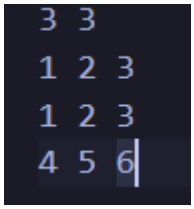
Запуск программы:



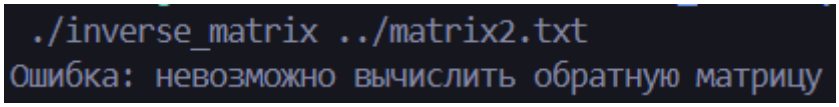
Ожидаемый результат, при вводе некорректных данных:



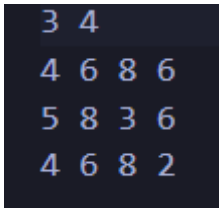
Файл matrix2.txt (вырожденная матрица, детерминант=0):



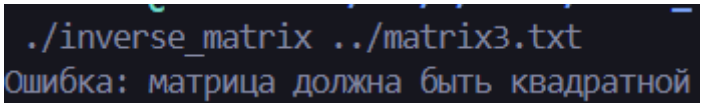
Результат:



Файл matrix3.txt (не квадратная):

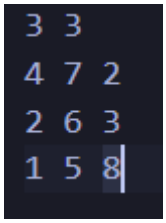


Результат:



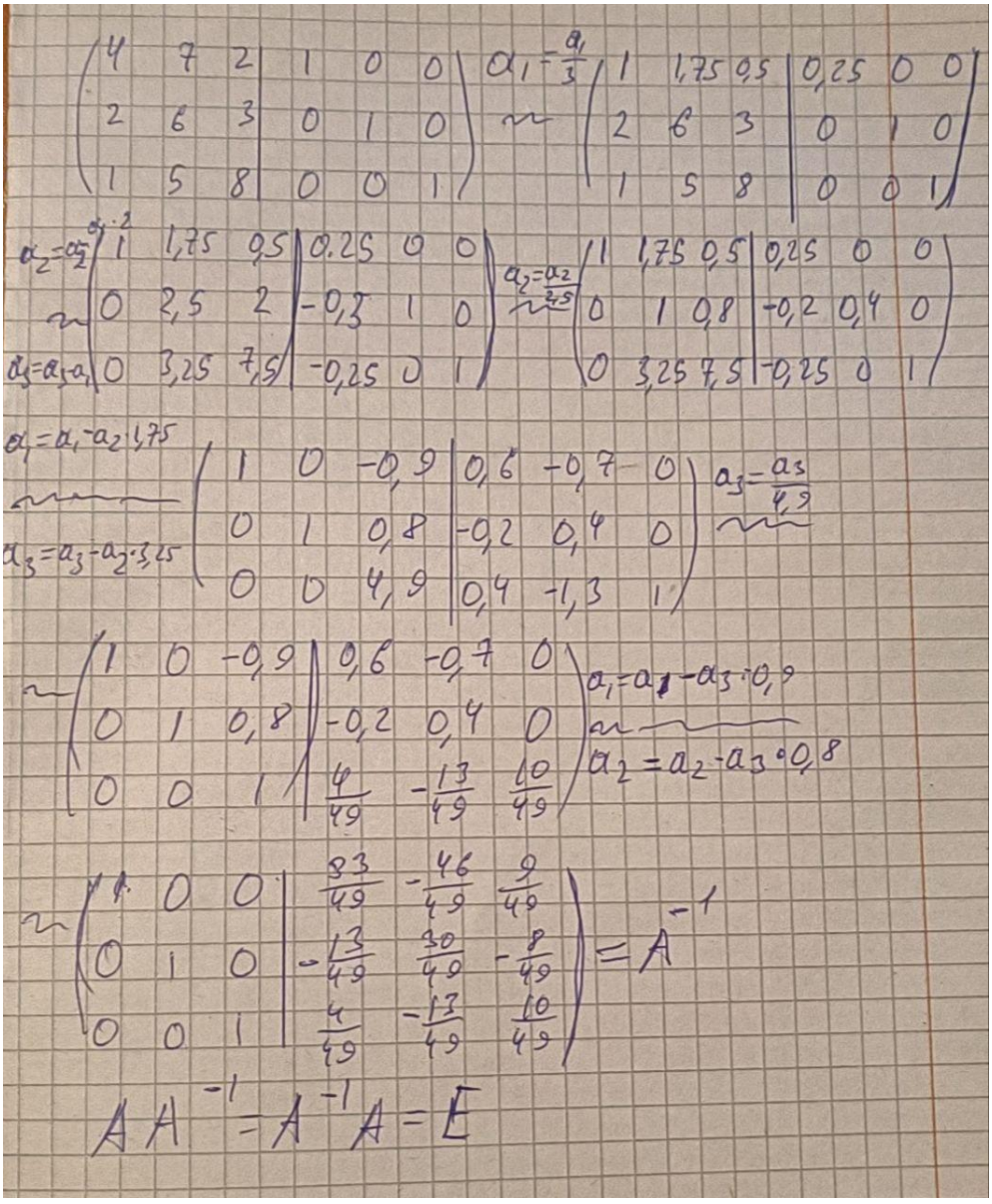
Корректный запуск программы:

Файл matrix.txt (исходный)



Результат:

```
 ./inverse_matrix ../matrix.txt
Исходная матрица:
  4.0000   7.0000   2.0000
  2.0000   6.0000   3.0000
  1.0000   5.0000   8.0000

Обратная матрица:
  0.6735  -0.9388   0.1837
 -0.2653   0.6122  -0.1633
  0.0816  -0.2653   0.2041

Результат умножения исходной матрицы на обратную:
  1.0000   0.0000  -0.0000
  0.0000   1.0000  -0.0000
 -0.0000   0.0000   1.0000

Проверка пройдена: произведение матриц равно единичной матрице
```



Используемые макросы CMocka:

assert_non_null() - Проверяет, что указатель не NULL

assert_null() - Проверяет, что указатель NULL

assert_int_equal() - Сравнивает целые числа

assert_true()/assert_false() - Проверка булевых значений

Пример использования:

assert_non_null(create_matrix(2, 2))

assert_null(create_matrix(-1, -1))

assert_int_equal(matrix->rows, 2)

assert_true(is_matrix_square(matrix))

Листинг Программы:

Файл matrix.h:

```c
#ifndef MATRIX_H
#define MATRIX_H
#include <math.h>

// Структура для хранения матрицы
typedef struct {
    double** data;
    int rows;
    int cols;
} Matrix;

// Функции для работы с матрицами
Matrix* create_matrix(int rows, int cols);
void free_matrix(Matrix* matrix);
Matrix* read_matrix_from_file(const char* filename);
void print_matrix(const Matrix* matrix);
int is_matrix_square(const Matrix* matrix);
Matrix* create_identity_matrix(int size);
int inverse_matrix(Matrix* matrix, Matrix* inverse);
Matrix* copy_matrix(const Matrix* matrix);
int multiply_matrices(const Matrix* a, const Matrix* b, Matrix* result);
int are_matrices_equal(const Matrix* a, const Matrix* b, double epsilon);

#endif
```

Файл matrix.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "matrix.h"
```

```c
// Создание матрицы
Matrix* create_matrix(int rows, int cols) {
    if (rows <= 0 || cols <= 0) {
        return NULL;
    }

    Matrix* matrix = (Matrix*)malloc(sizeof(Matrix));
    if (matrix == NULL) {
        return NULL;
    }

    matrix->rows = rows;
    matrix->cols = cols;

    matrix->data = (double**)malloc(rows * sizeof(double*));
    if (matrix->data == NULL) {
        free(matrix);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
        matrix->data[i] = (double*)malloc(cols * sizeof(double));
        if (matrix->data[i] == NULL) {
            for (int j = 0; j < i; j++) {
                free(matrix->data[j]);
            }
            free(matrix->data);
            free(matrix);
            return NULL;
        }
    }

    return matrix;
}

// Освобождение памяти матрицы
void free_matrix(Matrix* matrix) {
    if (matrix == NULL) {
        return;
    }

    for (int i = 0; i < matrix->rows; i++) {
        free(matrix->data[i]);
    }
    free(matrix->data);
    free(matrix);
}

// Чтение матрицы из файла
Matrix* read_matrix_from_file(const char* filename) {
    FILE* file = fopen(filename, "r");
```

```
    if (file == NULL) {
        printf("Ошибка: не удалось открыть файл %s\n", filename);
        return NULL;
    }

    int rows, cols;
    if (fscanf(file, "%d %d", &rows, &cols) != 2) {
        printf("Ошибка: неверный формат файла\n");
        fclose(file);
        return NULL;
    }

    Matrix* matrix = create_matrix(rows, cols);
    if (matrix == NULL) {
        printf("Ошибка: не удалось создать матрицу\n");
        fclose(file);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (fscanf(file, "%lf", &matrix->data[i][j]) != 1) {
                printf("Ошибка: неверные данные в файле\n");
                free_matrix(matrix);
                fclose(file);
                return NULL;
            }
        }
    }

    fclose(file);
    return matrix;
}

// Печать матрицы
void print_matrix(const Matrix* matrix) {
    if (matrix == NULL) {
        printf("Матрица не существует\n");
        return;
    }

    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            printf("%8.4f ", matrix->data[i][j]);
        }
        printf("\n");
    }
}

// Проверка, является ли матрица квадратной
int is_matrix_square(const Matrix* matrix) {
```

```c
    if (matrix == NULL) {
        return 0;
    }
    return matrix->rows == matrix->cols;
}

// Создание единичной матрицы
Matrix* create_identity_matrix(int size) {
    Matrix* matrix = create_matrix(size, size);
    if (matrix == NULL) {
        return NULL;
    }

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix->data[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }

    return matrix;
}

// Копирование матрицы
Matrix* copy_matrix(const Matrix* matrix) {
    if (matrix == NULL) {
        return NULL;
    }

    Matrix* copy = create_matrix(matrix->rows, matrix->cols);
    if (copy == NULL) {
        return NULL;
    }

    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            copy->data[i][j] = matrix->data[i][j];
        }
    }

    return copy;
}

// Вычисление обратной матрицы методом Гаусса
int inverse_matrix(Matrix* matrix, Matrix* inverse) {
    if (matrix == NULL || inverse == NULL || matrix->rows != matrix->cols ||
        inverse->rows != inverse->cols || matrix->rows != inverse->rows) {
        return 0;
    }

    int n = matrix->rows;
    double temp;
```

```
    // Присоединяем единичную матрицу справа
    Matrix* augmented = create_matrix(n, 2 * n);
    if (augmented == NULL) {
        return 0;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            augmented->data[i][j] = matrix->data[i][j];
            augmented->data[i][j + n] = (i == j) ? 1.0 : 0.0;
        }
    }

    //  метод Гаусса
    for (int i = 0; i < n; i++) {
        // Если элемент на диагонали равен 0, ищем строку для замены
        if (fabs(augmented->data[i][i]) < 1e-10) {
            int swap_row = -1;
            for (int k = i + 1; k < n; k++) {
                if (fabs(augmented->data[k][i]) > 1e-10) {
                    swap_row = k;
                    break;
                }
            }

            if (swap_row == -1) {
                free_matrix(augmented);
                return 0; // Матрица вырожденная(квадратная матрица, которая не
имеет обратной матрицы.)
            }

            // Меняем строки местами
            for (int j = 0; j < 2 * n; j++) {
                temp = augmented->data[i][j];
                augmented->data[i][j] = augmented->data[swap_row][j];
                augmented->data[swap_row][j] = temp;
            }
        }

        // Нормализуем текущую строку
        temp = augmented->data[i][i];
        for (int j = 0; j < 2 * n; j++) {
            augmented->data[i][j] /= temp;
        }

        // Обнуляем элементы ниже текущего
        for (int k = i + 1; k < n; k++) {
            temp = augmented->data[k][i];
            for (int j = 0; j < 2 * n; j++) {
                augmented->data[k][j] -= augmented->data[i][j] * temp;
```

```
            }
        }
    }

    // Обратный ход метода Гаусса
    for (int i = n - 1; i >= 0; i--) {
        for (int k = i - 1; k >= 0; k--) {
            temp = augmented->data[k][i];
            for (int j = 0; j < 2 * n; j++) {
                augmented->data[k][j] -= augmented->data[i][j] * temp;
            }
        }
    }

    // Копируем обратную матрицу из расширенной
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            inverse->data[i][j] = augmented->data[i][j + n];
        }
    }

    free_matrix(augmented);
    return 1;
}

// Умножение матриц
int multiply_matrices(const Matrix* a, const Matrix* b, Matrix* result) {
    if (a == NULL || b == NULL || result == NULL ||
        a->cols != b->rows || result->rows != a->rows || result->cols != b->cols)
{
        return 0;
    }

    for (int i = 0; i < a->rows; i++) {
        for (int j = 0; j < b->cols; j++) {
            result->data[i][j] = 0.0;
            for (int k = 0; k < a->cols; k++) {
                result->data[i][j] += a->data[i][k] * b->data[k][j];
            }
        }
    }

    return 1;
}

// Проверка равенства матриц с заданной точностью
int are_matrices_equal(const Matrix* a, const Matrix* b, double epsilon) {
    if (a == NULL || b == NULL || a->rows != b->rows || a->cols != b->cols) {
        return 0;
    }
```

```
    for (int i = 0; i < a->rows; i++) {
        for (int j = 0; j < a->cols; j++) {
            if (fabs(a->data[i][j] - b->data[i][j]) > epsilon) {
                return 0;
            }
        }
    }

    return 1;
}
```

Файл CMakeLists.txt (для проверки)

```
# Поиск библиотеки cmocka
find_library(CMOCKA_LIBRARY cmocka)
find_path(CMOCKA_INCLUDE_DIR cmocka.h)

if(CMOCKA_LIBRARY AND CMOCKA_INCLUDE_DIR)
    message(STATUS "Found cmocka: ${CMOCKA_LIBRARY}")

    # Создаем тестовый исполняемый файл
    add_executable(test_matrix
        test_matrix.c
        ../src/matrix.c
    )

    # Подключаем заголовочные файлы
    target_include_directories(test_matrix PRIVATE ../include
${CMOCKA_INCLUDE_DIR})

    # Подключаем cmocka
    target_link_libraries(test_matrix ${CMOCKA_LIBRARY})

    # Добавляем тест
    add_test(NAME matrix_tests COMMAND test_matrix)
else()
    message(WARNING "cmocka не найден, тестирование нельзя сделать")
endif()
```

Файл test_matrix.c

```
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include "matrix.h"
#include <math.h>

// Тест создания и освобождения матрицы
static void test_create_and_free_matrix(void** state) {
    Matrix* matrix = create_matrix(3, 3);
    assert_non_null(matrix);
    assert_int_equal(matrix->rows, 3);
```

```
    assert_int_equal(matrix->cols, 3);

    free_matrix(matrix);
}

// Тест проверки квадратной матрицы
static void test_is_matrix_square(void** state) {
    Matrix* square = create_matrix(2, 2);
    assert_non_null(square);
    assert_true(is_matrix_square(square));

    Matrix* non_square = create_matrix(2, 3);
    assert_non_null(non_square);
    assert_false(is_matrix_square(non_square));

    free_matrix(square);
    free_matrix(non_square);
}

// Тест создания единичной матрицы
static void test_create_identity_matrix(void** state) {
    Matrix* identity = create_identity_matrix(3);
    assert_non_null(identity);

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (i == j) {
                assert_true(fabs(identity->data[i][j] - 1.0) < 1e-10);
            } else {
                assert_true(fabs(identity->data[i][j]) < 1e-10);
            }
        }
    }

    free_matrix(identity);
}

// Тест копирования матрицы
static void test_copy_matrix(void** state) {
    Matrix* original = create_matrix(2, 2);
    assert_non_null(original);

    original->data[0][0] = 1.0;
    original->data[0][1] = 2.0;
    original->data[1][0] = 3.0;
    original->data[1][1] = 4.0;

    Matrix* copy = copy_matrix(original);
    assert_non_null(copy);

    assert_true(are_matrices_equal(original, copy, 1e-10));
```

```
    free_matrix(original);
    free_matrix(copy);
}

// Тест вычисления обратной матрицы
static void test_inverse_matrix(void** state) {
    Matrix* matrix = create_matrix(2, 2);
    assert_non_null(matrix);

    matrix->data[0][0] = 4.0;
    matrix->data[0][1] = 7.0;
    matrix->data[1][0] = 2.0;
    matrix->data[1][1] = 6.0;

    Matrix* inverse = create_matrix(2, 2);
    assert_non_null(inverse);

    assert_true(inverse_matrix(matrix, inverse));

    // Ожидаемая обратная матрица
    Matrix* expected_inverse = create_matrix(2, 2);
    assert_non_null(expected_inverse);

    expected_inverse->data[0][0] = 0.6;
    expected_inverse->data[0][1] = -0.7;
    expected_inverse->data[1][0] = -0.2;
    expected_inverse->data[1][1] = 0.4;

    assert_true(are_matrices_equal(inverse, expected_inverse, 1e-6));

    // Проверка, что произведение матрицы на обратную дает единичную матрицу
    Matrix* identity = create_matrix(2, 2);
    assert_non_null(identity);

    assert_true(multiply_matrices(matrix, inverse, identity));

    Matrix* true_identity = create_identity_matrix(2);
    assert_non_null(true_identity);

    assert_true(are_matrices_equal(identity, true_identity, 1e-6));

    free_matrix(matrix);
    free_matrix(inverse);
    free_matrix(expected_inverse);
    free_matrix(identity);
    free_matrix(true_identity);
}

// Тест для вырожденной матрицы (не должна иметь обратной)
static void test_singular_matrix(void** state) {
```

```
    Matrix* matrix = create_matrix(2, 2);
    assert_non_null(matrix);

    matrix->data[0][0] = 1.0;
    matrix->data[0][1] = 2.0;
    matrix->data[1][0] = 2.0;
    matrix->data[1][1] = 4.0;

    Matrix* inverse = create_matrix(2, 2);
    assert_non_null(inverse);

    assert_false(inverse_matrix(matrix, inverse));

    free_matrix(matrix);
    free_matrix(inverse);
}

int main(void) {
    const struct CMUnitTest tests[] = {
        cmocka_unit_test(test_create_and_free_matrix),
        cmocka_unit_test(test_is_matrix_square),
        cmocka_unit_test(test_create_identity_matrix),
        cmocka_unit_test(test_copy_matrix),
        cmocka_unit_test(test_inverse_matrix),
        cmocka_unit_test(test_singular_matrix),
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}
```

Файл CMakeLists.txt (для создания основной программы, обратной матрицы)

```
cmake_minimum_required(VERSION 3.10)
project(inverse_matrix C)

set(CMAKE_C_STANDARD 11)
set(CMAKE_C_STANDARD_REQUIRED ON)

# Основная программа
add_executable(inverse_matrix
    src/matrix.c
    main.c
)

# Подключаем заголовочные файлы
target_include_directories(inverse_matrix PRIVATE include)

enable_testing()

add_subdirectory(test)
```

Файл main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Использование: %s <имя_файла_с_матрицей>\n", argv[0]);
        return 1;
    }

    Matrix* matrix = read_matrix_from_file(argv[1]);
    if (matrix == NULL) {
        return 1;
    }

    // Проверка, что матрица квадратная
    if (!is_matrix_square(matrix)) {
        printf("Ошибка: матрица должна быть квадратной\n");
        free_matrix(matrix);
        return 1;
    }

    // Создание матрицы для обратной матрицы
    Matrix* inverse = create_matrix(matrix->rows, matrix->cols);
    if (inverse == NULL) {
        printf("Ошибка: не удалось создать матрицу для обратной\n");
        free_matrix(matrix);
        return 1;
    }

    if (!inverse_matrix(matrix, inverse)) {
        printf("Ошибка: невозможно вычислить обратную матрицу \n");
        free_matrix(matrix);
        free_matrix(inverse);
        return 1;
    }

    printf("Исходная матрица:\n");
    print_matrix(matrix);

    printf("\nОбратная матрица:\n");
    print_matrix(inverse);

    // Проверка: умножение исходной матрицы на обратную должно дать единичную
матрицу
    Matrix* identity = create_matrix(matrix->rows, matrix->cols);
    if (identity == NULL) {
        printf("Ошибка: не удалось создать матрицу для проверки\n");
        free_matrix(matrix);
```

```
        free_matrix(inverse);
        return 1;
    }

    if (!multiply_matrices(matrix, inverse, identity)) {
        printf("Ошибка: не удалось проверить результат\n");
        free_matrix(matrix);
        free_matrix(inverse);
        free_matrix(identity);
        return 1;
    }

    Matrix* true_identity = create_identity_matrix(matrix->rows);
    if (true_identity == NULL) {
        printf("Ошибка: не удалось создать единичную матрицу для проверки\n");
        free_matrix(matrix);
        free_matrix(inverse);
        free_matrix(identity);
        return 1;
    }

    printf("\nРезультат умножения исходной матрицы на обратную:\n");
    print_matrix(identity);

    if (are_matrices_equal(identity, true_identity, 1e-6)) {
        printf("\nПроверка пройдена: произведение матриц равно единичной
матрице\n");
    } else {
        printf("\nПредупреждение: произведение матриц не точно равно единичной
матрице\n");
    }

    // Освобождение памяти
    free_matrix(matrix);
    free_matrix(inverse);
    free_matrix(identity);
    free_matrix(true_identity);

    return 0;
}
```

Файл matrix.txt

3 3

4 7 2

2 6 3

1 5 8

Файл matrix1.txt (ошибка ввода)

3 3

q qwe tr

sdf h fg

sdf sdf sdf

Файл matrix2.txt (детерминант=0)

3 3

1 2 3

1 2 3

4 5 6

Файл matrix3.txt (не квадратная матрица)

3 4

4 6 8 6

5 8 3 6

4 6 8 2

Условия для создания обратной матрицы:

1. **Квадратная матрица**
2. **Детерминант (определитель) матрицы не должен быть равен нулю**

Список литературы:

1. Видео Bing
2. Нахождение решения СЛУ и обратной матрицы методом Гаусса-Жордана
3. C-Programming/2024-2025/sw_testing/tests/test_example.c at master · kruffka/C-Programming
4. Юнит тесты на Си — нет ничего проще / Хабр
5. cmocka_unit_testing_and_mocking.pdf
6. Как вычислить обратную матрицу? Пошаговый алгоритм с примерами