

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Расчетно-графическая работа
по дисциплине «Программирование»

Студент:

Группа ИКС-433

А.И Бубенина

Преподаватель:

А.И. Вейлер

Новосибирск 2025

1 ВЫЧИСЛЕНИЕ ОБРАТНОЙ МАТРИЦЫ

1.1 Постановка задачи

Программа должна вычислять обратную матрицу для заданной квадратной матрицы. Входные данные поступают из файла, имя которого передается как аргумент командной строки. Программа выполняет:

- Проверку корректности входных данных
- Вычисление обратной матрицы методом Гаусса-Жордана
- Верификацию результата через умножение на исходную матрицу
- Вывод результата или сообщения об ошибке

1.2 Критерии оценки

- **Хорошо:**
 - Проверка размера матрицы
 - Верификация результата
 - Динамическое выделение памяти
- **Отлично** (дополнительно):
 - Сборка через CMake

1.3 Реализация

1.3.1 Алгоритм

Основной алгоритм - метод Гаусса-Жордана:

1. Формирование расширенной матрицы $[A|I]$
2. Прямой ход: приведение к верхнетреугольному виду
3. Обратный ход: приведение к единичной матрице
4. Извлечение обратной матрицы из правой части

1.3.2 Структура программы

Программа состоит из следующих модулей:

- `matrix.h` - заголовочный файл с объявлениями функций
- `matrix.c` - реализация операций с матрицами
- `main.c` - основной код программы
- `test.matrix.txt` - модульное тестирование
- `CMakeLists.txt` - конфигурация сборки

1.4 Тестирование

```
arina@DESKTOP-JV410DG:~/lab8/build$ ./matrix_tests ../matrix.txt
[=====] Running 2 test(s).
[ RUN      ] test_inverse_matrix_correct
[      OK   ] test_inverse_matrix_correct
[ RUN      ] test_inverse_matrix_incorrect
[      OK   ] test_inverse_matrix_incorrect
[=====] 2 test(s) run.
[ PASSED   ] 2 test(s).
```

Рисунок 1 — Модульное тестирование с корректными и некорректными входными данными

```
arina@DESKTOP-JV410DG:~/lab8/build$ ./matrix_app ../matrix.txt
Исходная матрица (5x5):
  2      3      1      7      8
 -4      6      7      2      1
  9      3      4      5      1
  7     -3      1      0      9
  2      1      7      6     -5

Обратная матрица:
-0.043 -0.024  0.111  0.009 -0.035
-0.002  0.072  0.139 -0.098 -0.137
-0.062  0.084 -0.038  0.093  0.076
 0.120 -0.074 -0.045 -0.044  0.089
 0.039  0.033 -0.035  0.061 -0.027

Проверка результата (A * A-1):
  1      0      0      0      0
  0      1      0      0      0
  0      0      1      0      0
  0      0      0      1      0
  0      0      0      0      1
arina@DESKTOP-JV410DG:~/lab8/build$
```

Рисунок 2 — Запуск основного файла

1.5 Исходный код

Основные компоненты программы:

1.5.1 main.c

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Ошибка: %s не введено имя файла\n", argv[0]);
        return 1;
    }

    int n;
    double **matrix = read_matrix(argv[1], &n);
    if (!matrix) {
        return 1;
    }

    printf("Исходная матрица (%dx%d):\n", n, n);
    print_matrix_isx(n, matrix);

    int error;
    double **inverse = inverse_matrix(n, matrix, &error);

    if (error) {
        printf("\nОбратной матрицы не существует.\n");
    } else {
        printf("\nОбратная матрица:\n");
        print_matrix(n, inverse);

        verify_inverse(n, matrix, inverse);
    }

    free_matrix(n, matrix);
    if (inverse) free_matrix(n, inverse);
}
```

```
    return 0;
}
```

1.5.2 matrix.c

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

#define EPSILON 1e-10 // для учета погрешности вычислений
#define MAX_SIZE 100

// Объявления функций
double** memory_matrix(int n, int m);
void free_matrix(int n, double **matrix);
double** read_matrix(const char *filename, int *n);
void print_matrix(int n, double **matrix);
double** inverse_matrix(int n, double **matrix, int *error);
void verify_inverse(int n, double **A, double **A_inv);

// Функция для выделения памяти под матрицу
double** memory_matrix(int n, int m) {
    double **matrix = (double**)malloc(n * sizeof(double*));
    for (int i = 0; i < n; i++) {
        matrix[i] = (double*)malloc(m * sizeof(double));
    }
    return matrix;
}

// Функция для освобождения памяти матрицы
void free_matrix(int n, double **matrix) {
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
}
```

```

    }
    free(matrix);
}

// Функция для чтения матрицы из файла
double** read_matrix(const char *filename, int *n) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Ошибка: не удалось открыть файл %s\n", filename);
        return NULL;
    }

    char line[4096];
    *n = 0;
    int cols = 0;
    double **matrix = memory_matrix(MAX_SIZE, MAX_SIZE);

    while (fgets(line, sizeof(line), file) && *n < MAX_SIZE) {
        char *token = strtok(line, " \t\n"); // Разбиение на токены
        int current_cols = 0;

        while (token != NULL && current_cols < MAX_SIZE) {
            if (*n == 0) {
                matrix[*n][current_cols] = atof(token); // Преобразование
            } else {
                if (current_cols >= cols) {
                    fprintf(stderr, "Ошибка: матрица не квадратная\n");
                    fclose(file);
                    free_matrix(MAX_SIZE, matrix);
                    return NULL;
                }
                matrix[*n][current_cols] = atof(token);
            }
            current_cols++;
            token = strtok(NULL, " \t\n");
        }
    }
}

```

```

    }

    if (*n == 0) {
        cols = current_cols;
    } else if (current_cols != cols) {
        fprintf(stderr, "Ошибка: матрица не квадратная\n");
        fclose(file);
        free_matrix(MAX_SIZE, matrix);
        return NULL;
    }

    (*n)++;
}

fclose(file);

if (*n != cols) {
    fprintf(stderr, "Ошибка: матрица не квадратная (%dx%d)\n", *n,
        free_matrix(MAX_SIZE, matrix);
    return NULL;
}

// Создаем матрицу нужного размера
double **result = memory_matrix(*n, *n);
for (int i = 0; i < *n; i++) {
    for (int j = 0; j < *n; j++) {
        result[i][j] = matrix[i][j];
    }
}

free_matrix(MAX_SIZE, matrix);
return result;
}

// Функция для вывода матрицы (округление до 3 знаков)

```

```

void print_matrix(int n, double **matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%8.3f ", matrix[i][j]);
        }
        printf("\n");
    }
}

// Функция для вывода матрицы (без округления)
void print_matrix_isx(int n, double **matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%8.0f ", matrix[i][j]);
        }
        printf("\n");
    }
}

// Функция для вычисления обратной матрицы методом Гаусса-Жордана
double** inverse_matrix(int n, double **matrix, int *error) {
    *error = 0;

    // Создаем расширенную матрицу [A|I]
    double **augmented = memory_matrix(n, 2*n);
    double **inverse = memory_matrix(n, n);

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

    // Прямой ход метода Гаусса
    for (int col = 0; col < n; col++) {

```



```

// Поиск строки с максимальным элементом
int max_row = col;
for (int i = col + 1; i < n; i++) {
    if (fabs(augmented[i][col]) > fabs(augmented[max_row][col])
        max_row = i;
    }
}

// Проверка
if (fabs(augmented[max_row][col]) < EPSILON) {
    *error = 1;
    free_matrix(n, augmented);
    free_matrix(n, inverse);
    return NULL;
}

// Перестановка строк
if (max_row != col) {
    for (int j = 0; j < 2*n; j++) {
        double temp = augmented[col][j];
        augmented[col][j] = augmented[max_row][j];
        augmented[max_row][j] = temp;
    }
}

// Нормализация текущей строки
double pivot = augmented[col][col];
for (int j = 0; j < 2*n; j++) {
    augmented[col][j] /= pivot;
}

// Обнуление других строк
for (int i = 0; i < n; i++) {
    if (i != col) {
        double factor = augmented[i][col];

```

```

        for (int j = 0; j < 2*n; j++) {
            augmented[i][j] -= factor * augmented[col][j];
        }
    }
}

// Извлечение обратной матрицы
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        inverse[i][j] = augmented[i][j+n];
    }
}

free_matrix(n, augmented);
return inverse;
}

// Функция для проверки результата ( $A * A^{-1} = I$ )
void verify_inverse(int n, double **A, double **A_inv) {
    double **product = memory_matrix(n, n);

    printf("\nПроверка результата ( $A * A^{-1}$ ):\n");

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            product[i][j] = 0.0;
            for (int k = 0; k < n; k++) {
                product[i][j] += A[i][k] * A_inv[k][j];
            }
            // Округляем до 2 знаков для вывода
            printf("%8.0f ", fabs(product[i][j] - (i == j ? 1.0 : 0.0))
                (i == j ? 1.0 : 0.0) : product[i][j]));
        }
        printf("\n");
    }
}

```

```

    }

    free_matrix(n, product);
}

```

1.5.3 test.matrix.c

```

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

// Тест с корректными входными данными (матрица должна иметь обратную)
static void test_inverse_matrix_correct(void **state) {
    (void)state; // Неиспользуемый параметр

    int n = 2;
    int error = 0;

    // Создаем тестовую матрицу 2x2
    double **matrix = memory_matrix(n, n);
    matrix[0][0] = 4; matrix[0][1] = 7;
    matrix[1][0] = 2; matrix[1][1] = 6;

    // Вычисляем обратную матрицу
    double **inverse = inverse_matrix(n, matrix, &error);

    // Проверяем, что ошибки нет
    assert_int_equal(error, 0);

    // Проверяем правильность вычисления (ожидаемая обратная матрица)
    double expected[2][2] = {

```

```

        {0.6, -0.7},
        {-0.2, 0.4}
    };

    // Сравниваем с ожидаемым результатом с учетом погрешности
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            assert_true(fabs(inverse[i][j] - expected[i][j]) < EPSILON)
        }
    }

    // Освобождаем память
    free_matrix(n, matrix);
    free_matrix(n, inverse);
}

// Тест с некорректными входными данными (матрица не должна иметь обратную)
static void test_inverse_matrix_incorrect(void **state) {
    (void)state; // Неиспользуемый параметр

    int n = 2;
    int error = 0;

    // Создаем вырожденную матрицу 2x2 (определитель = 0)
    double **matrix = memory_matrix(n, n);
    matrix[0][0] = 1; matrix[0][1] = 2;
    matrix[1][0] = 2; matrix[1][1] = 4;

    // Пытаемся вычислить обратную матрицу
    double **inverse = inverse_matrix(n, matrix, &error);

    // Проверяем, что функция вернула ошибку
    assert_int_equal(error, 1);
    assert_null(inverse);
}

```

```

    // Освобождаем память
    free_matrix(n, matrix);
}

// Основная функция для запуска тестов
int main(void) {
    const struct CUnitTest tests[] = {
        cmocka_unit_test(test_inverse_matrix_correct),
        cmocka_unit_test(test_inverse_matrix_incorrect),
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}

```

1.5.4 matrix.h

```

#ifndef MATRIX_H
#define MATRIX_H

#define EPSILON 1e-10
#define MAX_SIZE 100

double** memory_matrix(int n, int m);
void free_matrix(int n, double **matrix);
double** read_matrix(const char *filename, int *n);
void print_matrix(int n, double **matrix);
void print_matrix_isx(int n, double **matrix);
double** inverse_matrix(int n, double **matrix, int *error);
void verify_inverse(int n, double **A, double **A_inv);

#endif

```

1.5.5 CMakeLists.txt

```

cmake_minimum_required(VERSION 3.10)

```

```

project(MatrixInversion C)

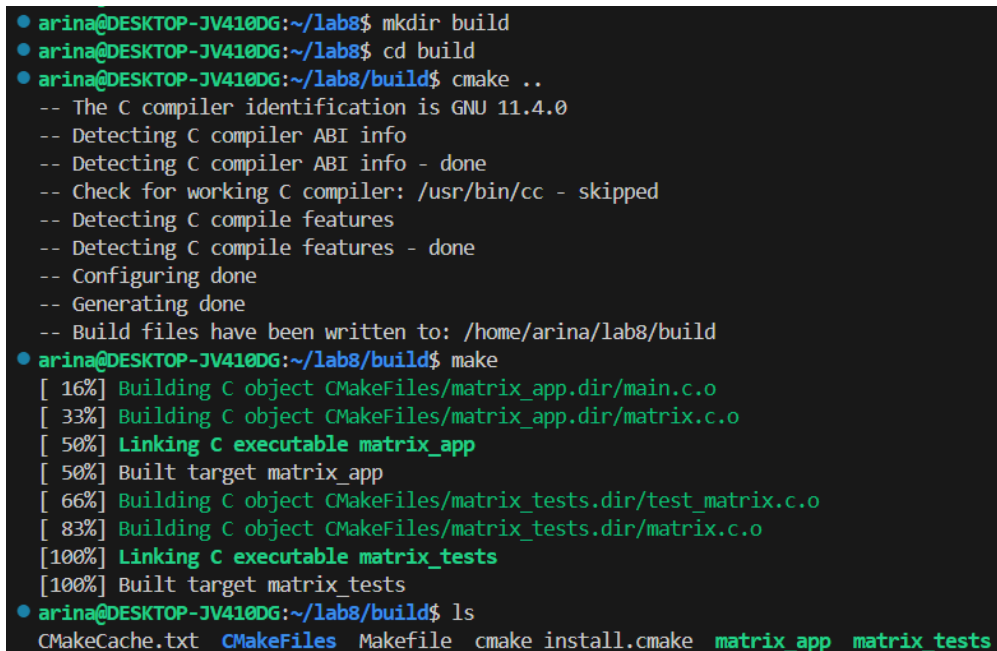
set(CMAKE_C_STANDARD 99)

add_executable(matrix_app
    main.c
    matrix.c
)

# Тесты
find_package(cmocka REQUIRED)
add_executable(matrix_tests
    test_matrix.c
    matrix.c
)

target_link_libraries(matrix_tests cmocka)

```



```

● arina@DESKTOP-JV410DG:~/lab8$ mkdir build
● arina@DESKTOP-JV410DG:~/lab8$ cd build
● arina@DESKTOP-JV410DG:~/lab8/build$ cmake ..
-- The C compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/arina/lab8/build
● arina@DESKTOP-JV410DG:~/lab8/build$ make
[ 16%] Building C object CMakeFiles/matrix_app.dir/main.c.o
[ 33%] Building C object CMakeFiles/matrix_app.dir/matrix.c.o
[ 50%] Linking C executable matrix_app
[ 50%] Built target matrix_app
[ 66%] Building C object CMakeFiles/matrix_tests.dir/test_matrix.c.o
[ 83%] Building C object CMakeFiles/matrix_tests.dir/matrix.c.o
[100%] Linking C executable matrix_tests
[100%] Built target matrix_tests
● arina@DESKTOP-JV410DG:~/lab8/build$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  matrix_app  matrix_tests

```

Рисунок 3 — Процесс сборки через CMake

1.6 Заключение

Программа успешно реализует:

- Корректное вычисление обратных матриц
- Проверку входных данных
- Предварительную проверку размера матрицы
- Верификацию результатов
- Эффективное управление памятью

Сборка через CMake обеспечивает переносимость и удобство развёртывания.