Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

**ОТЧЕТ**
по практической работе 1

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИС-541
«17» февраля 2026 г. _____ /Даниленко К.А./

Проверил:
Ст. преподаватель кафедры ВС
«28»февраля 2026 г. _____ /Карнилов А.А./

Оценка «_____»

Новосибирск 2026

# ОГЛАВЛЕНИЕ

ЗАДАНИЕ ..... 3
ВЫПОЛНЕНИЕ РАБОТЫ ..... 4
ПРИЛОЖЕНИЕ ..... 5

# ЗАДАНИЕ

*Реализовать структуру (struct) в соответствии с вариантом. Структура должна обеспечивать набор методов для работы с данными размещая объекты в динамической памяти компьютера. Структура должна быть описана с использованием typedef. Создать методы: конструктор, конструктор "копирования", конструктор "по умолчанию". Реализовать указанные методы с динамическим выделением памяти для хранения перечисленных в соответствии с вариантом полей. Внимательно рассмотреть цикл работы с динамической памяти и обратить внимание на вложенность. Создать метод деструктор для освобождения памяти. Посмотреть, как вызываются конструкторы и деструкторы. Реализовать структуру отдельным модулем (struct.h, struct.c). Где struct название структуры. Собрать демонстрационную программу. Для сборки демонстрационной программы использовать makefile.*

*Вариант 2. Двумерная матрица (matrix2d):*

*Задание на 3.*

*Создайте структуру matrix2d, представляющую двумерную матрицу. Значения матрицы могут быть представлены как целыми, так и вещественными значениями не более чем 32bit. Матрицы могут быть с главной диагональю и без нее.*

*Реализуйте методы для:*

- *Сравнения двух матриц (==, !=, >, =, <=)*
- *Ввода/Вывода матриц*
- *Изменение данных структуры (++, --, setter's).*
- *Заполнение матрицы случайными значениями*

*Задание на 4.*

- *Получение столбца/строки*
- *Транспонирование матрицы*
- *Нахождение определителя (детерминант)*

*Задание на 5.*

- *Вычисление обратной матрицы*

# ВЫПОЛНЕНИЕ РАБОТЫ

Описывается ход работы над заданием с приложением снимков экрана;

Создал заголовочный и исполняемые файлы. В заголовочном файле прописал функции, необходимые библиотеки и константы.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX_MATRICES 10
#define EPSILON 0.000001f
```

В файле main.c реализовано меню, которое работает посредством вызова функций.

```
matrix2d *matrices[MAX_MATRICES] = {NULL};
    int matrix_count = 0;
    char choice;

    printf("ПРАКТИЧЕСКАЯ РАБОТА №1 (Вариант 2)\n");
    printf("Структура: Двумерная матрица (matrix2d)\n");

    do
    {
        printf("\nМЕНЮ:\n");
        printf("1. Создать новую матрицу\n");
        printf("2. Показать все матрицы\n");
        printf("3. Выбрать матрицу для операций\n");
        printf("4. Сравнить две матрицы\n");
        printf("5. Уничтожить матрицу\n");
        printf("6. Уничтожить все матрицы\n");
        printf("0. Выход\n");

        printf("Текущее количество матриц: %d/%d\n", matrix_count, MAX_MATRICES);

        printf("Выберите действие: ");
```

В файле matrix2d.c реализован функционал функций для работы с матрицами.

```
void menu_destroy(matrix2d **m)
{
    if (!*m)
    {
        printf("Матрица не создана!\n");
        return;
    }

    matrix2d_destroy(*m);
    *m = NULL;
    printf("Матрица уничтожена.\n");
}
```

# ПРИЛОЖЕНИЕ

Исходный код с комментариями;

**matrix2d.h – заголовочный файл содержащий библиотеки, константы и заготовки функций**

```
#ifndef MATRIX2D_H
#define MATRIX2D_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX_MATRICES 10
#define EPSILON 0.000001f

// Структура двумерной матрицы
typedef struct
{
    int rows;     // Количество строк
    int cols;     // Количество столбцов
    int has_diag; // Флаг: имеет ли матрица главную диагональ (1 - да, 0 - нет)
    float **data; // Динамический массив указателей на строки
} matrix2d;

// Функция для очистки буфера ввода
void clear();

// Конструкторы и деструктор
matrix2d *matrix2d_default();
matrix2d *matrix2d_create(int rows, int cols);
matrix2d *matrix2d_copy(const matrix2d *src);
void matrix2d_destroy(matrix2d *m);

// Операторы сравнения
int matrix2d_eq(const matrix2d *m1, const matrix2d *m2);
int matrix2d_ne(const matrix2d *m1, const matrix2d *m2);
int matrix2d_lt(const matrix2d *m1, const matrix2d *m2);
int matrix2d_gt(const matrix2d *m1, const matrix2d *m2);
int matrix2d_le(const matrix2d *m1, const matrix2d *m2);
int matrix2d_ge(const matrix2d *m1, const matrix2d *m2);

// Ввод/вывод матриц
void matrix2d_input(matrix2d *m);
void matrix2d_print(const matrix2d *m);

// Изменение данных структуры
void matrix2d_set(matrix2d *m, int row, int col, float value);
float matrix2d_get(const matrix2d *m, int row, int col);
void matrix2d_inc(matrix2d *m);
void matrix2d_dec(matrix2d *m);

// Заполнение матрицы случайными значениями
void matrix2d_fill_random(matrix2d *m, float min, float max);
```

```
// Получение столбца/строки
matrix2d *matrix2d_get_row(const matrix2d *m, int row_idx);
matrix2d *matrix2d_get_col(const matrix2d *m, int col_idx);

// Транспонирование матрицы
matrix2d *matrix2d_transpose(const matrix2d *m);

// Нахождение определителя (детерминант)
float matrix2d_determinant(const matrix2d *m);

// Вычисление обратной матрицы
matrix2d *matrix2d_inverse(const matrix2d *m);

// Функции меню
void menu_create_matrix(matrix2d **m);
void menu_print_matrix(const matrix2d *m);
void menu_fill_random(matrix2d *m);
void menu_set_element(matrix2d *m);
void menu_increment(matrix2d *m);
void menu_decrement(matrix2d *m);
void menu_get_row(const matrix2d *m);
void menu_get_col(const matrix2d *m);
void menu_transpose(matrix2d **m);
void menu_determinant(const matrix2d *m);
void menu_inverse(const matrix2d *m);
void menu_compare(const matrix2d *m1, const matrix2d *m2);
void menu_destroy(matrix2d **m);

// Вспомогательные функции
int is_matrix_created(const matrix2d *m);
float matrix2d_norm(const matrix2d *m);

#endif
```

**main.c – исполняемый файл с меню вызова функций**

```
#include "matrix2d.h"

int main()
{
    matrix2d *matrices[MAX_MATRICES] = {NULL};
    int matrix_count = 0;
    char choice;

    printf("ПРАКТИЧЕСКАЯ РАБОТА №1 (Вариант 2)\n");
    printf("Структура: Двумерная матрица (matrix2d)\n");

    do
    {
        printf("\nМЕНЮ:\n");
        printf("1. Создать новую матрицу\n");
        printf("2. Показать все матрицы\n");
        printf("3. Выбрать матрицу для операций\n");
        printf("4. Сравнить две матрицы\n");
        printf("5. Уничтожить матрицу\n");
        printf("6. Уничтожить все матрицы\n");
        printf("0. Выход\n");
```

```
printf("Текущее количество матриц: %d/%d\n", matrix_count, MAX_MATRICES);

printf("Выберите действие: ");
if (scanf(" %c", &choice) == EOF)
{
    printf("\nОбнаружен конец ввода (EOF). Завершение программы...\n");
    printf("До свидания!\n");

    // Освобождаем всю память
    for (int i = 0; i < MAX_MATRICES; i++)
    {
        if (matrices[i])
        {
            menu_destroy(&matrices[i]);
        }
    }
    return 0;
}
clear();

switch (choice)
{
case '1':
    if (matrix_count >= MAX_MATRICES)
    {
        printf("Достигнуто максимальное количество матриц (%d)!\n", MAX_MATRICES);
        break;
    }

    // Находим первый свободный слот
    int free_index = -1;
    for (int i = 0; i < MAX_MATRICES; i++)
    {
        if (matrices[i] == NULL)
        {
            free_index = i;
            break;
        }
    }

    if (free_index != -1)
    {
        printf("\nСоздание матрицы с индексом %d:\n", free_index + 1);
        menu_create_matrix(&matrices[free_index]);
        if (matrices[free_index])
        {
            matrix_count++;
            printf("Матрица %d создана успешно!\n", free_index + 1);
        }
    }
    break;

case '2':
    printf("\n--- СПИСОК ВСЕХ МАТРИЦ ---\n");
    int found = 0;
    for (int i = 0; i < MAX_MATRICES; i++)
```

```
            {
                if (matrices[i])
                {
                    found = 1;
                    printf("\nМатрица %d:\n", i + 1);
                    matrix2d_print(matrices[i]);
                }
            }
            if (!found)
            {
                printf("Нет созданных матриц.\n");
            }
            break;

        case '3':
            if (matrix_count == 0)
            {
                printf("Нет созданных матриц!\n");
                break;
            }

            printf("\n--- ВЫБОР МАТРИЦЫ ДЛЯ ОПЕРАЦИЙ ---\n");
            printf("Доступные матрицы: ");
            for (int i = 0; i < MAX_MATRICES; i++)
            {
                if (matrices[i])
                {
                    printf("%d ", i + 1);
                }
            }
            printf("\n");

            int matrix_index;
            printf("Введите номер матрицы: ");
            if (scanf("%d", &matrix_index) != 1)
            {
                printf("Ошибка ввода!\n");
                clear();
                break;
            }
            clear();

            if (matrix_index < 1 || matrix_index > MAX_MATRICES || matrices[matrix_index -
1] == NULL)
            {
                printf("Неверный номер матрицы!\n");
                break;
            }

            matrix_index--; // Переводим в 0-индексацию

            // Подменю для выбранной матрицы
            char sub_choice;
            do
            {
                printf("\n--- ОПЕРАЦИИ С МАТРИЦЕЙ %d ---\n", matrix_index + 1);
                printf("a. Вывести матрицу\n");
```

```
        printf("b. Заполнить случайными значениями\n");
        printf("c. Изменить элемент матрицы\n");
        printf("d. Инкремент матрицы (++)\n");
        printf("e. Декремент матрицы (--)\n");
        printf("f. Получить строку матрицы\n");
        printf("g. Получить столбец матрицы\n");
        printf("h. Транспонировать матрицу\n");
        printf("i. Определитель матрицы (если квадратная)\n");
        printf("j. Обратная матрица (если квадратная)\n");
        printf("r. Вернуться в главное меню\n");

        printf("Выберите операцию: ");
        if (scanf(" %c", &sub_choice) == EOF)
        {
            sub_choice = 'r';
        }
        clear();

        switch (sub_choice)
        {
        case 'a':
            menu_print_matrix(matrices[matrix_index]);
            break;
        case 'b':
            menu_fill_random(matrices[matrix_index]);
            break;
        case 'c':
            menu_set_element(matrices[matrix_index]);
            break;
        case 'd':
            menu_increment(matrices[matrix_index]);
            break;
        case 'e':
            menu_decrement(matrices[matrix_index]);
            break;
        case 'f':
            menu_get_row(matrices[matrix_index]);
            break;
        case 'g':
            menu_get_col(matrices[matrix_index]);
            break;
        case 'h':
            menu_transpose(&matrices[matrix_index]);
            break;
        case 'i':
            menu_determinant(matrices[matrix_index]);
            break;
        case 'j':
            menu_inverse(matrices[matrix_index]);
            break;
        case 'r':
            printf("Возврат в главное меню...\n");
            break;
        default:
            printf("Неверный выбор!\n");
        }
```

```
            if (sub_choice != 'r')
            {
                printf("\nНажмите Enter для продолжения...");
                getchar();
            }

        } while (sub_choice != 'r');
        break;

    case '4':
        if (matrix_count < 2)
        {
            printf("Недостаточно матриц для сравнения! Создайте хотя бы 2 матрицы.\n");
            break;
        }

        printf("\n--- СРАВНЕНИЕ ДВУХ МАТРИЦ ---\n");
        printf("Доступные матрицы: ");
        for (int i = 0; i < MAX_MATRICES; i++)
        {
            if (matrices[i])
            {
                printf("%d ", i + 1);
            }
        }
        printf("\n");

        int m1_idx, m2_idx;
        printf("Введите номер первой матрицы: ");
        if (scanf("%d", &m1_idx) != 1)
        {
            printf("Ошибка ввода!\n");
            clear();
            break;
        }
        clear();

        printf("Введите номер второй матрицы: ");
        if (scanf("%d", &m2_idx) != 1)
        {
            printf("Ошибка ввода!\n");
            clear();
            break;
        }
        clear();

        if (m1_idx < 1 || m1_idx > MAX_MATRICES || matrices[m1_idx - 1] == NULL)
        {
            printf("Неверный номер первой матрицы!\n");
            break;
        }
        if (m2_idx < 1 || m2_idx > MAX_MATRICES || matrices[m2_idx - 1] == NULL)
        {
            printf("Неверный номер второй матрицы!\n");
            break;
        }
```

```
            menu_compare(matrices[m1_idx - 1], matrices[m2_idx - 1]);
            break;

        case '5':
            if (matrix_count == 0)
            {
                printf("Нет созданных матриц!\n");
                break;
            }

            printf("\n--- УНИЧТОЖЕНИЕ МАТРИЦЫ ---\n");
            printf("Доступные матрицы: ");
            for (int i = 0; i < MAX_MATRICES; i++)
            {
                if (matrices[i])
                {
                    printf("%d ", i + 1);
                }
            }
            printf("\n");

            int del_idx;
            printf("Введите номер матрицы для уничтожения: ");
            if (scanf("%d", &del_idx) != 1)
            {
                printf("Ошибка ввода!\n");
                clear();
                break;
            }
            clear();

            if (del_idx < 1 || del_idx > MAX_MATRICES || matrices[del_idx - 1] == NULL)
            {
                printf("Неверный номер матрицы!\n");
                break;
            }

            menu_destroy(&matrices[del_idx - 1]);
            matrix_count--;
            printf("Матрица %d уничтожена.\n", del_idx);
            break;

        case '6':
            if (matrix_count == 0)
            {
                printf("Нет созданных матриц!\n");
                break;
            }

            printf("Уничтожение всех %d матриц...\n", matrix_count);
            for (int i = 0; i < MAX_MATRICES; i++)
            {
                if (matrices[i])
                {
                    menu_destroy(&matrices[i]);
                }
            }
```

```
            matrix_count = 0;
            printf("Все матрицы уничтожены.\n");
            break;

        case '0':
            printf("До свидания!\n");
            break;

        default:
            printf("Неверный выбор!\n");
        }

        if (choice != '0')
        {
            printf("\nНажмите Enter для продолжения...");
            getchar();
        }

    } while (choice != '0');

    return 0;
}
```

**matrix2d.c – основная логика программы**

```
#include "matrix2d.h"

// Функция для очистки буфера ввода
void clear()
{
    int c;
    while ((c = getchar()) != '\n' && c != EOF)
    {
    }
}

// Вспомогательная функция: абсолютное значение float
static float _fabs(float x)
{
    return (x < 0.0f) ? -x : x;
}

// Проверка, создана ли матрица
int is_matrix_created(const matrix2d *m)
{
    if (!m)
    {
        printf("Матрица не создана!\n");
        return 0;
    }
    return 1;
}

// Норма Фробениуса (квадрат нормы - без корня)
float matrix2d_norm(const matrix2d *m)
{
    if (!m)
```

```
        return 0.0f;
    float sum = 0.0f;
    for (int i = 0; i < m->rows; i++)
    {
        for (int j = 0; j < m->cols; j++)
        {
            float val = m->data[i][j];
            sum += val * val;
        }
    }
    return sum;
}

matrix2d *matrix2d_default()
{
    return matrix2d_create(1, 1);
}

matrix2d *matrix2d_create(int rows, int cols)
{
    if (rows <= 0 || cols <= 0)
    {
        printf("Ошибка: размеры матрицы должны быть положительными!\n");
        return NULL;
    }

    matrix2d *m = (matrix2d *)malloc(sizeof(matrix2d));
    if (!m)
    {
        printf("Ошибка выделения памяти!\n");
        return NULL;
    }

    m->rows = rows;
    m->cols = cols;
    m->has_diag = (rows == cols) ? 1 : 0;

    m->data = (float **)malloc(rows * sizeof(float *));
    if (!m->data)
    {
        free(m);
        printf("Ошибка выделения памяти!\n");
        return NULL;
    }

    for (int i = 0; i < rows; i++)
    {
        m->data[i] = (float *)calloc(cols, sizeof(float));
        if (!m->data[i])
        {
            for (int j = 0; j < i; j++)
            {
                free(m->data[j]);
            }
            free(m->data);
            free(m);
            printf("Ошибка выделения памяти!\n");
```

```
            return NULL;
        }
    }

    return m;
}

matrix2d *matrix2d_copy(const matrix2d *src)
{
    if (!src)
    {
        printf("Ошибка: исходная матрица не существует!\n");
        return NULL;
    }

    matrix2d *m = matrix2d_create(src->rows, src->cols);
    if (!m)
        return NULL;

    for (int i = 0; i < src->rows; i++)
    {
        memcpy(m->data[i], src->data[i], src->cols * sizeof(float));
    }

    return m;
}

void matrix2d_destroy(matrix2d *m)
{
    if (!m)
        return;

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

int matrix2d_eq(const matrix2d *m1, const matrix2d *m2)
{
    if (!m1 || !m2)
        return 0;
    if (m1->rows != m2->rows || m1->cols != m2->cols)
        return 0;

    for (int i = 0; i < m1->rows; i++)
    {
        for (int j = 0; j < m1->cols; j++)
        {
            if (_fabs(m1->data[i][j] - m2->data[i][j]) > EPSILON)
            {
                return 0;
```

```
            }
        }
    }
    return 1;
}

int matrix2d_ne(const matrix2d *m1, const matrix2d *m2)
{
    return !matrix2d_eq(m1, m2);
}

int matrix2d_lt(const matrix2d *m1, const matrix2d *m2)
{
    float norm1 = matrix2d_norm(m1);
    float norm2 = matrix2d_norm(m2);
    return (norm1 < norm2 - EPSILON) ? 1 : 0;
}

int matrix2d_gt(const matrix2d *m1, const matrix2d *m2)
{
    float norm1 = matrix2d_norm(m1);
    float norm2 = matrix2d_norm(m2);
    return (norm1 > norm2 + EPSILON) ? 1 : 0;
}

int matrix2d_le(const matrix2d *m1, const matrix2d *m2)
{
    return !matrix2d_gt(m1, m2);
}

int matrix2d_ge(const matrix2d *m1, const matrix2d *m2)
{
    return !matrix2d_lt(m1, m2);
}

void matrix2d_input(matrix2d *m)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return;
    }

    printf("Введите %d элементов матрицы %dx%d:\n", m->rows * m->cols, m->rows, m->cols);
    for (int i = 0; i < m->rows; i++)
    {
        for (int j = 0; j < m->cols; j++)
        {
            printf("  [%d][%d] = ", i, j);
            if (scanf("%f", &m->data[i][j]) != 1)
            {
                printf("Ошибка ввода!\n");
                clear();
                return;
            }
        }
    }
```

```
}

void matrix2d_print(const matrix2d *m)
{
    if (!m)
    {
        printf("[NULL matrix]\n");
        return;
    }

    printf("Матрица %dx%d [диагональ %s]:\n",
           m->rows, m->cols, m->has_diag ? "есть" : "нет");

    for (int i = 0; i < m->rows; i++)
    {
        printf("[");
        for (int j = 0; j < m->cols; j++)
        {
            printf("%8.3f", m->data[i][j]);
            if (j < m->cols - 1)
                printf(" ");
        }
        printf("]\n");
    }
}

void matrix2d_set(matrix2d *m, int row, int col, float value)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return;
    }
    if (row < 0 || row >= m->rows || col < 0 || col >= m->cols)
    {
        printf("Ошибка: индекс вне диапазона!\n");
        return;
    }
    m->data[row][col] = value;
}

float matrix2d_get(const matrix2d *m, int row, int col)
{
    if (!m)
        return 0.0f;
    if (row < 0 || row >= m->rows || col < 0 || col >= m->cols)
    {
        printf("Ошибка: индекс вне диапазона!\n");
        return 0.0f;
    }
    return m->data[row][col];
}

void matrix2d_inc(matrix2d *m)
{
    if (!m)
    {
```

```
        printf("Ошибка: матрица не создана!\n");
        return;
    }
    for (int i = 0; i < m->rows; i++)
    {
        for (int j = 0; j < m->cols; j++)
        {
            m->data[i][j] += 1.0f;
        }
    }
}

void matrix2d_dec(matrix2d *m)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return;
    }
    for (int i = 0; i < m->rows; i++)
    {
        for (int j = 0; j < m->cols; j++)
        {
            m->data[i][j] -= 1.0f;
        }
    }
}

void matrix2d_fill_random(matrix2d *m, float min, float max)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return;
    }

    static int seeded = 0;
    if (!seeded)
    {
        srand((unsigned int)time(NULL));
        seeded = 1;
    }

    float range = max - min;
    for (int i = 0; i < m->rows; i++)
    {
        for (int j = 0; j < m->cols; j++)
        {
            m->data[i][j] = min + range * (float)rand() / (RAND_MAX + 1.0f);
        }
    }
}

static matrix2d *_matrix2d_minor(const matrix2d *m, int skip_row, int skip_col)
{
    if (!m || !m->has_diag || m->rows <= 1)
        return NULL;
```

```
    int n = m->rows - 1;
    matrix2d *minor = matrix2d_create(n, n);
    if (!minor)
        return NULL;

    int di = 0;
    for (int i = 0; i < m->rows; i++)
    {
        if (i == skip_row)
            continue;
        int dj = 0;
        for (int j = 0; j < m->cols; j++)
        {
            if (j == skip_col)
                continue;
            minor->data[di][dj++] = m->data[i][j];
        }
        di++;
    }

    return minor;
}

matrix2d *matrix2d_get_row(const matrix2d *m, int row_idx)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return NULL;
    }
    if (row_idx < 0 || row_idx >= m->rows)
    {
        printf("Ошибка: индекс строки вне диапазона!\n");
        return NULL;
    }

    matrix2d *row = matrix2d_create(1, m->cols);
    if (!row)
        return NULL;

    memcpy(row->data[0], m->data[row_idx], m->cols * sizeof(float));
    return row;
}

matrix2d *matrix2d_get_col(const matrix2d *m, int col_idx)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return NULL;
    }
    if (col_idx < 0 || col_idx >= m->cols)
    {
        printf("Ошибка: индекс столбца вне диапазона!\n");
        return NULL;
    }
```

```
    matrix2d *col = matrix2d_create(m->rows, 1);
    if (!col)
        return NULL;

    for (int i = 0; i < m->rows; i++)
    {
        col->data[i][0] = m->data[i][col_idx];
    }
    return col;
}

matrix2d *matrix2d_transpose(const matrix2d *m)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return NULL;
    }

    matrix2d *t = matrix2d_create(m->cols, m->rows);
    if (!t)
        return NULL;

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

    return t;
}

float matrix2d_determinant(const matrix2d *m)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return 0.0f;
    }
    if (!m->has_diag)
    {
        printf("Ошибка: определитель можно найти только для квадратной матрицы!\n");
        return 0.0f;
    }

    int n = m->rows;

    if (n == 1)
        return m->data[0][0];
    if (n == 2)
    {
        return m->data[0][0] * m->data[1][1] - m->data[0][1] * m->data[1][0];
    }
```

```
    float det = 0.0f;
    for (int j = 0; j < n; j++)
    {
        matrix2d *minor = _matrix2d_minor(m, 0, j);
        if (minor)
        {
            float subdet = matrix2d_determinant(minor);
            det += ((j % 2) ? -1.0f : 1.0f) * m->data[0][j] * subdet;
            matrix2d_destroy(minor);
        }
    }

    return det;
}

matrix2d *matrix2d_inverse(const matrix2d *m)
{
    if (!m)
    {
        printf("Ошибка: матрица не создана!\n");
        return NULL;
    }
    if (!m->has_diag)
    {
        printf("Ошибка: обратную матрицу можно найти только для квадратной матрицы!\n");
        return NULL;
    }

    float det = matrix2d_determinant(m);
    if (_fabs(det) < EPSILON)
    {
        printf("Ошибка: матрица вырожденная, обратная не существует!\n");
        return NULL;
    }

    int n = m->rows;
    matrix2d *inv = matrix2d_create(n, n);
    if (!inv)
        return NULL;

    if (n == 1)
    {
        inv->data[0][0] = 1.0f / m->data[0][0];
        return inv;
    }
    if (n == 2)
    {
        inv->data[0][0] = m->data[1][1] / det;
        inv->data[0][1] = -m->data[0][1] / det;
        inv->data[1][0] = -m->data[1][0] / det;
        inv->data[1][1] = m->data[0][0] / det;
        return inv;
    }

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

```
        {
            matrix2d *minor = _matrix2d_minor(m, i, j);
            if (minor)
            {
                float subdet = matrix2d_determinant(minor);
                inv->data[j][i] = (((i + j) % 2) ? -1.0f : 1.0f) * subdet / det;
                matrix2d_destroy(minor);
            }
        }
    }

    return inv;
}

void menu_create_matrix(matrix2d **m)
{
    if (*m)
    {
        printf("Матрица уже существует! Уничтожьте её сначала.\n");
        return;
    }

    int rows, cols;
    printf("\nСоздание матрицы:\n");
    printf("Введите количество строк: ");
    if (scanf("%d", &rows) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    printf("Введите количество столбцов: ");
    if (scanf("%d", &cols) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (rows <= 0 || cols <= 0)
    {
        printf("Ошибка: размеры должны быть положительными!\n");
        return;
    }

    *m = matrix2d_create(rows, cols);
    if (!*m)
    {
        printf("Ошибка создания матрицы!\n");
        return;
    }

    int choice;
    printf("Хотите ввести данные вручную? (1 - да, 0 - нет): ");
```

```
    if (scanf("%d", &choice) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (choice == 1)
    {
        matrix2d_input(*m);
    }

    printf("Матрица успешно создана!\n");
    matrix2d_print(*m);
}

void menu_print_matrix(const matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    printf("\nТекущая матрица:\n");
    matrix2d_print(m);
}

void menu_fill_random(matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    float min, max;
    printf("\nЗаполнение случайными значениями:\n");
    printf("Введите минимальное значение: ");
    if (scanf("%f", &min) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    printf("Введите максимальное значение: ");
    if (scanf("%f", &max) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (min >= max)
    {
        printf("Ошибка: минимальное значение должно быть меньше максимального!\n");
        return;
    }
```

```
    matrix2d_fill_random(m, min, max);
    printf("Матрица заполнена случайными значениями!\n");
    matrix2d_print(m);
}

void menu_set_element(matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    int row, col;
    float value;

    printf("\nИзменение элемента матрицы:\n");
    printf("Текущая матрица:\n");
    matrix2d_print(m);

    printf("Введите номер строки (0-%d): ", m->rows - 1);
    if (scanf("%d", &row) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    printf("Введите номер столбца (0-%d): ", m->cols - 1);
    if (scanf("%d", &col) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (row < 0 || row >= m->rows || col < 0 || col >= m->cols)
    {
        printf("Ошибка: индекс вне диапазона!\n");
        return;
    }

    printf("Введите новое значение: ");
    if (scanf("%f", &value) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    matrix2d_set(m, row, col, value);
    printf("Элемент [%d][%d] изменен на %.3f!\n", row, col, value);
    matrix2d_print(m);
}

void menu_increment(matrix2d *m)
{
```

```
    if (!is_matrix_created(m))
        return;

    printf("\nИнкремент матрицы (++): все элементы + 1\n");
    printf("До:\n");
    matrix2d_print(m);

    matrix2d_inc(m);

    printf("После:\n");
    matrix2d_print(m);
    printf("Инкремент выполнен!\n");
}

void menu_decrement(matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    printf("\nДекремент матрицы (--): все элементы - 1\n");
    printf("До:\n");
    matrix2d_print(m);

    matrix2d_dec(m);

    printf("После:\n");
    matrix2d_print(m);
    printf("Декремент выполнен!\n");
}

void menu_get_row(const matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    int row_idx;
    printf("\nПолучение строки матрицы:\n");
    printf("Введите номер строки (0-%d): ", m->rows - 1);
    if (scanf("%d", &row_idx) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (row_idx < 0 || row_idx >= m->rows)
    {
        printf("Ошибка: индекс строки вне диапазона!\n");
        return;
    }

    matrix2d *row = matrix2d_get_row(m, row_idx);
    if (!row)
    {
        printf("Ошибка получения строки!\n");
        return;
```

```
    }

    printf("Строка %d:\n", row_idx);
    matrix2d_print(row);
    matrix2d_destroy(row);
}

void menu_get_col(const matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    int col_idx;
    printf("\nПолучение столбца матрицы:\n");
    printf("Введите номер столбца (0-%d): ", m->cols - 1);
    if (scanf("%d", &col_idx) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (col_idx < 0 || col_idx >= m->cols)
    {
        printf("Ошибка: индекс столбца вне диапазона!\n");
        return;
    }

    matrix2d *col = matrix2d_get_col(m, col_idx);
    if (!col)
    {
        printf("Ошибка получения столбца!\n");
        return;
    }

    printf("Столбец %d:\n", col_idx);
    matrix2d_print(col);
    matrix2d_destroy(col);
}

void menu_transpose(matrix2d **m)
{
    if (!is_matrix_created(*m))
        return;

    printf("\nТранспонирование матрицы:\n");
    printf("Исходная:\n");
    matrix2d_print(*m);

    matrix2d *transposed = matrix2d_transpose(*m);
    if (!transposed)
    {
        printf("Ошибка транспонирования!\n");
        return;
    }
```

```
    printf("Транспонированная:\n");
    matrix2d_print(transposed);

    int choice;
    printf("Заменить исходную матрицу? (1 - да, 0 - нет): ");
    if (scanf("%d", &choice) != 1)
    {
        printf("Ошибка ввода!\n");
        clear();
        return;
    }
    clear();

    if (choice == 1)
    {
        matrix2d_destroy(*m);
        *m = transposed;
        printf("Матрица заменена!\n");
    }
    else
    {
        matrix2d_destroy(transposed);
        printf("Матрица осталась без изменений.\n");
    }
}

void menu_determinant(const matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    if (!m->has_diag)
    {
        printf("Ошибка: определитель можно найти только для квадратной матрицы!\n");
        printf("Текущая матрица: %dx%d\n", m->rows, m->cols);
        return;
    }

    printf("\nВычисление определителя:\n");
    matrix2d_print(m);

    float det = matrix2d_determinant(m);
    printf("Определитель det = %.6f\n", det);
}

void menu_inverse(const matrix2d *m)
{
    if (!is_matrix_created(m))
        return;

    if (!m->has_diag)
    {
        printf("Ошибка: обратную матрицу можно найти только для квадратной матрицы!\n");
        printf("Текущая матрица: %dx%d\n", m->rows, m->cols);
        return;
    }
```

```
    printf("\nВычисление обратной матрицы:\n");
    printf("Исходная:\n");
    matrix2d_print(m);

    float det = matrix2d_determinant(m);
    printf("Определитель: %.6f\n", det);

    if (_fabs(det) < 1e-6f)
    {
        printf("Матрица вырожденная - обратная не существует!\n");
        return;
    }

    matrix2d *inverse = matrix2d_inverse(m);
    if (!inverse)
    {
        printf("Ошибка вычисления!\n");
        return;
    }

    printf("Обратная матрица:\n");
    matrix2d_print(inverse);

    float det_inv = matrix2d_determinant(inverse);
    printf("Проверка: det(A) * det(A^-1) = %.6f\n", det * det_inv);

    matrix2d_destroy(inverse);
}

void menu_compare(const matrix2d *m1, const matrix2d *m2)
{
    if (!is_matrix_created(m1))
    {
        printf("Матрица A не создана!\n");
        return;
    }
    if (!is_matrix_created(m2))
    {
        printf("Матрица B не создана!\n");
        return;
    }

    printf("\nСравнение матриц A и B:\n");
    printf("Матрица A:\n");
    matrix2d_print(m1);

    printf("Матрица B:\n");
    matrix2d_print(m2);

    printf("\nРезультаты сравнения:\n");
    printf("A == B: %s\n", matrix2d_eq(m1, m2) ? "true" : "false");
    printf("A != B: %s\n", matrix2d_ne(m1, m2) ? "true" : "false");
    printf("A > B:  %s\n", matrix2d_gt(m1, m2) ? "true" : "false");
    printf("A < B:  %s\n", matrix2d_lt(m1, m2) ? "true" : "false");
    printf("A >= B: %s\n", matrix2d_ge(m1, m2) ? "true" : "false");
    printf("A <= B: %s\n", matrix2d_le(m1, m2) ? "true" : "false");
}
```

```
void menu_destroy(matrix2d **m)
{
    if (!*m)
    {
        printf("Матрица не создана!\n");
        return;
    }

    matrix2d_destroy(*m);
    *m = NULL;
    printf("Матрица уничтожена.\n");
}
```