МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Кафедра телекоммуникационных систем и вычислительных средств (ТС и ВС)

РГР по дисциплине «Программирование»

по теме: «Вычисление обратной матрицы»

Студент: Группа ИКС-432 Салий В.П.

Преподаватель: Старший Преподаватель Вейлер А.И.

Новосибирск 2025 г

2. Задание

Разработать программу на языке C, которая:

1. Считывает квадратную матрицу из файла.
2. Проверяет, существует ли для неё обратная матрица (определитель ≠ 0).
3. Вычисляет обратную матрицу методом алгебраических дополнений.
4. Выводит результат в консоль и проверяет его корректность (умножение исходной матрицы на обратную должно дать единичную матрицу).
5. Обрабатывает ошибки (некорректный ввод, не квадратная матрица, нулевой определитель).

Требования:

- Динамическое выделение памяти.
- Проверка входных данных.
- Сборка через CMake.

3. Анализ задачи

Методы и алгоритмы

1. **Проверка квадратности матрицы**
    - Сравниваем количество строк и столбцов.
2. **Вычисление определителя**
    - Рекурсивный метод разложения по первой строке.
    - Для матрицы 1×1: det = a[0][0].
    - Для матрицы 2×2: det = a[0][0] * a[1][1] - a[0][1] * a[1][0].
    - Для матриц n×n:

Copy

Download

```
det = 0
для каждого элемента в первой строке:
   minor = подматрица без текущей строки и столбца
   det += элемент * (-1)^(i+j) * det(minor)
```

3. **Поиск обратной матрицы**
    - Вычисляем матрицу алгебраических дополнений.
    - Транспонируем её (получаем присоединённую матрицу).
    - Делим каждый элемент на определитель исходной матрицы.

Псевдокод

```
Функция inverse_matrix(матрица A):
    если A не квадратная:
        вернуть ошибку
    det = determinant(A)
    если det == 0:
        вернуть "Обратной матрицы не существует"
    cofactor = матрица алгебраических дополнений(A)
    adjugate = транспонировать(cofactor)
    inverse = adjugate / det
    вернуть inverse
```

4. Тестовые данные

Корректные данные

Файл: matrix1.txt

```
2 5
1 3
```

Ожидаемый результат:

```
Обратная матрица:
 3.0 -5.0
-1.0  2.0
```

Некорректные данные

1. **Не квадратная матрица**
   Файл: invalid1.txt

```
1 2 3
```

4 5 6

Ожидаемый вывод:
Ошибка: матрица не квадратная

2. **Нулевой определитель**
   Файл: singular.txt

1 2

2 4

Ожидаемый вывод:
Ошибка: определитель равен 0, обратной матрицы не существует

5. Скриншоты с результатами

6. Листинг программы

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <matrix_file>\n", argv[0]);
        return 1;
    }
    Matrix *matrix = read_matrix_from_file(argv[1]);
    if (matrix == NULL) {
```

```
    printf("Error reading matrix from file\n");
    return 1;
}
printf("Original matrix:\n");
print_matrix(matrix);
if (matrix->rows != matrix->cols) {
    printf("Matrix is not square, cannot compute inverse\n");
    free_matrix(matrix);
    return 1;
}
double det = determinant(matrix);
if (det == 0) {
    printf("Matrix is singular (determinant is zero), inverse does not exist\n");
    free_matrix(matrix);
    return 1;
}
Matrix *inverse = inverse_matrix(matrix);
if (inverse == NULL) {
    printf("Error computing inverse matrix\n");
    free_matrix(matrix);
    return 1;
}
printf("\nInverse matrix:\n");
print_matrix(inverse);
// Verification
Matrix *identity = multiply_matrices(matrix, inverse);
if (identity != NULL) {
    printf("\nVerification (original * inverse):\n");
    print_matrix(identity);
    free_matrix(identity);
}
free_matrix(matrix);
```

```
    free_matrix(inverse);
    return 0;
}
```

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

typedef struct {
    int rows;
    int cols;
    double **data;
} Matrix;

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);
double determinant(const Matrix *matrix);
Matrix *inverse_matrix(const Matrix *matrix);
Matrix *multiply_matrices(const Matrix *a, const Matrix *b);

#endif // MATRIX_H
```

Matrix.c

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

Matrix *create_matrix(int rows, int cols) {
    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;
}

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);
}
```

```
Matrix *read_matrix_from_file(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return NULL;
    }

    int rows = 0, cols = 0;
    char line[1024];

    // First pass to determine dimensions
    while (fgets(line, sizeof(line), file) != NULL) {
        rows++;
        char *token = strtok(line, " \t\n");
        int current_cols = 0;
        while (token != NULL) {
            current_cols++;
            token = strtok(NULL, " \t\n");
        }
        if (rows == 1) {
            cols = current_cols;
        } else if (current_cols != cols) {
            printf("Inconsistent number of columns in row %d\n", rows);
            fclose(file);
            return NULL;
        }
    }

    if (rows == 0 || cols == 0) {
        fclose(file);
        return NULL;
```

```
    }

    // Rewind file for second pass to read data
    rewind(file);

    Matrix *matrix = create_matrix(rows, cols);
    if (matrix == NULL) {
        fclose(file);
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
        if (fgets(line, sizeof(line), file) != NULL) {
            char *token = strtok(line, " \t\n");
            for (int j = 0; j < cols && token != NULL; j++) {
                matrix->data[i][j] = atof(token);
                token = strtok(NULL, " \t\n");
            }
        }
    }

    fclose(file);
    return matrix;
}

void print_matrix(const Matrix *matrix) {
    if (matrix == NULL) 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");
```

```
    }
}

Matrix *create_submatrix(const Matrix *matrix, int exclude_row, int exclude_col) {
    if (matrix == NULL || matrix->rows <= 1 || matrix->cols <= 1) return NULL;

    Matrix *submatrix = create_matrix(matrix->rows - 1, matrix->cols - 1);
    if (submatrix == NULL) return NULL;

    int sub_i = 0;
    for (int i = 0; i < matrix->rows; i++) {
        if (i == exclude_row) continue;
        int sub_j = 0;
        for (int j = 0; j < matrix->cols; j++) {
            if (j == exclude_col) continue;
            submatrix->data[sub_i][sub_j] = matrix->data[i][j];
            sub_j++;
        }
        sub_i++;
    }

    return submatrix;
}

double determinant(const Matrix *matrix) {
    if (matrix == NULL || matrix->rows != matrix->cols) {
        return NAN;
    }

    if (matrix->rows == 1) {
        return matrix->data[0][0];
    }
```

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

    double det = 0;
    for (int j = 0; j < matrix->cols; j++) {
        Matrix *submatrix = create_submatrix(matrix, 0, j);
        if (submatrix == NULL) {
            return NAN;
        }
        double sub_det = determinant(submatrix);
        free_matrix(submatrix);
        det += matrix->data[0][j] * pow(-1, j) * sub_det;
    }

    return det;
}

Matrix *transpose_matrix(const Matrix *matrix) {
    if (matrix == NULL) return NULL;

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

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

```
    return transposed;
}

Matrix *cofactor_matrix(const Matrix *matrix) {
    if (matrix == NULL || matrix->rows != matrix->cols) return NULL;

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

    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            Matrix *submatrix = create_submatrix(matrix, i, j);
            if (submatrix == NULL) {
                free_matrix(cofactor);
                return NULL;
            }
            double sub_det = determinant(submatrix);
            free_matrix(submatrix);
            cofactor->data[i][j] = pow(-1, i + j) * sub_det;
        }
    }

    return cofactor;
}

Matrix *inverse_matrix(const Matrix *matrix) {
    if (matrix == NULL || matrix->rows != matrix->cols) return NULL;

    double det = determinant(matrix);
    if (det == 0) return NULL;

    Matrix *cofactor = cofactor_matrix(matrix);
```

```
    if (cofactor == NULL) return NULL;

    Matrix *adjugate = transpose_matrix(cofactor);
    free_matrix(cofactor);
    if (adjugate == NULL) return NULL;

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

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

    free_matrix(adjugate);
    return inverse;
}

Matrix *multiply_matrices(const Matrix *a, const Matrix *b) {
    if (a == NULL || b == NULL || a->cols != b->rows) return NULL;

    Matrix *result = create_matrix(a->rows, b->cols);
    if (result == NULL) return NULL;

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

```
                result->data[i][j] += a->data[i][k] * b->data[k][j];
            }
        }
    }

    return result;
}
```