МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №14

по дисциплине: Основы программирования
тема: «Работа с многомерными массивами»

Выполнил: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверили:
ст. преп. Притчин Иван Сергеевич
асс. Черников Сергей Викторович

Белгород 2023 г.

Лабораторная работа №14 «Работа с многомерными массивами»

Цель работы: получение навыков работы с многомерными массивами.

Содержание отчета:

Тема лабораторной работы

Цель лабораторной работы

Решения задач. Для каждой задачи указаны:

– Текст задачи.

– Исходный код.

Вывод.

Задача №1.

Дана квадратная матрица, все элементы которой различны. Поменять местами строки, в которых находятся максимальный и минимальный элементы.

Код программы:

```
#include <stdio.h>
#include <stdbool.h>
#include "libs/algorithms/matrix/matrix.h"

int main(void) {
    printf("Enter the size of the square matrix: ");

    long long size_a;
    scanf("%lld", &size_a);

    printf("Enter the elements of the matrix size of
%lldx%lld\n", size_a, size_a);

    Matrix a = get_matrix(size_a, size_a);
    input_matrix(a);

    Position max_value = get_max_value_pos(a);
    Position min_value = get_min_value_pos(a);

    swap_rows(a, max_value.row_pos, min_value.row_pos);

    printf("Result\n");
    output_matrix(a);

    free_matrix(a);

    return 0;
}
```

Задача №6.

Даны две квадратные матрицы A и B. Определить, являются ли они взаимно обратными ($A = B^{-1}$).

(a) `bool is_mutually_inverse_matrices(Matrix m1, Matrix m2)`

Код программы:

```
#include <stdio.h>
#include <stdbool.h>
#include "libs/algorithms/matrix/matrix.h"

// Возвращает единичную матрицу размера n x n
Matrix identity_matrix(long long n) {
    Matrix matrix;
    matrix.n_rows = n;
    matrix.n_cols = n;
    matrix.values = (long long**)malloc(n * sizeof(long long*));

    for (long long i = 0; i < n; i++) {
        matrix.values[i] = (long long*)malloc(n * sizeof(long
long));

        for (long long j = 0; j < n; j++)
            if (i == j)
                matrix.values[i][j] = 1;
            else
                matrix.values[i][j] = 0;
    }

    return matrix;
}
```

```
// Возвращает обратную матрицу к матрице matrix
Matrix invert_matrix(Matrix matrix) {
    Matrix identity = identity_matrix(matrix.n_rows);
    Matrix augmented = get_matrix(matrix.n_rows, 2 *
matrix.n_cols);

    for (long long i = 0; i < matrix.n_rows; i++)
        for (long long j = 0; j < matrix.n_cols; j++) {
            augmented.values[i][j] = matrix.values[i][j];
            augmented.values[i][j + matrix.n_cols] =
identity.values[i][j];
        }

    for (long long i = 0; i < matrix.n_rows; i++) {
        long long pivot_row = i;

        while (augmented.values[pivot_row][i] == 0) {
            pivot_row++;

            if (pivot_row == matrix.n_rows) {
                printf("Matrix is not invertible\n");

                free_matrix(identity);
                free_matrix(augmented);

                return get_matrix(0, 0);
            }
        }

        if (pivot_row != i) {
            long long *temp_row = augmented.values[i];

            augmented.values[i] = augmented.values[pivot_row];
            augmented.values[pivot_row] = temp_row;
        }

        long long pivot = augmented.values[i][i];

        for (long long j = 0; j < 2 * matrix.n_cols; j++)
            augmented.values[i][j] /= pivot;

        for (long long j = 0; j < matrix.n_rows; j++)
            if (j != i) {
                long long factor = augmented.values[j][i];

                for (long long k = 0; k < 2 * matrix.n_cols;
k++)
                    augmented.values[j][k] -= factor *
augmented.values[i][k];
```

```
            }
        }

        Matrix inverse = get_matrix(matrix.n_rows, matrix.n_cols);

        for (long long i = 0; i < matrix.n_rows; i++)
            for (long long j = 0; j < matrix.n_cols; j++)
                inverse.values[i][j] = augmented.values[i][j +
matrix.n_cols];

        free_matrix(identity);
        free_matrix(augmented);

        return inverse;
}


bool is_mutually_inverse_matrices(Matrix m1, Matrix m2) {
        Matrix invert_m1 = get_matrix(m1.n_rows, m1.n_rows);
        invert m1 = invert matrix(m1);

        return are two matrices equal(invert m1, m2);
}
```

```
int main(void) {
    printf("Enter the size of the square matrix A: ");

    long long size_a;
    scanf("%lld", &size_a);

    printf("Enter the elements of the matrix A size of
%lldx%lld\n", size_a, size_a);

    Matrix a = get_matrix(size_a, size_a);
    input_matrix(a);

    printf("Enter the size of the square matrix B: ");

    long long size b;
    scanf("%lld", &size_b);

    printf("Enter the elements of the matrix B size of
%lldx%lld\n", size b, size b);

    Matrix b = get matrix(size b, size b);
    input_matrix(b);

    printf("Result: ");
    printf(is mutually inverse matrices(a, b) ? "A^(-1) == B" :
"A^(-1) != B");

    free_matrix(a);
    free_matrix(b);

    return 0;
}
```

Задача №7.

Дана прямоугольная матрица. Назовем псевдодиагональю множество элементов этой матрицы, лежащих на прямой, параллельной прямой, содержащей элементы $a_{i,i}$. Найти сумму максимальных элементов всех псевдодиагоналей данной матрицы.

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include "libs/algorithms/matrix/matrix.h"

long long max2(long long a, long long b) {
    return a > b ? a : b;
}

long long find_sum_of_maxes_of_pseudo_diagonal(Matrix m) {
    long long* max values = (long long*) malloc((m.n rows +
m.n_cols - 1) * sizeof(long long));

    for (long long i = 0; i < m.n_rows + m.n_cols - 1; i++)
        max_values[i] = LLONG_MIN;

    for (long long i = 0; i < m.n_rows; i++)
        for (long long j = 0; j < m.n_cols; j++) {
            long long pseudo diagonal index = i - j + m.n cols -
1;

            max_values[pseudo_diagonal_index] =
max2(max values[pseudo diagonal index], m.values[i][j]);
        }

    long long sum_of_maxes = 0;

    for (long long i = 0; i < m.n_rows + m.n_cols - 1; i++)
        sum_of_maxes += max_values[i];

    free(max_values);

    return sum_of_maxes;
}
```

```
int main(void) {
    printf("Enter the size of the square matrix\n");

    printf("N (rows): ");
    long long n;
    scanf("%lld", &n);

    printf("M (cols): ");
    long long m;
    scanf("%lld", &m);

    printf("Enter the elements of the matrix size of
%lldx%lld\n", n, m);

    Matrix matrix = get matrix(n, m);
    input_matrix(matrix);

    long long sum_of_maxes =
find sum of maxes of pseudo diagonal(matrix);
    printf("Result: %lld", sum_of_maxes);

    free_matrix(matrix);

    return 0;
}
```

Задача №10.

Определить количество классов эквивалентных строк данной прямоугольной матрицы. Строки считать эквивалентными, если равны суммы их элементов.

(a) `int cmp_long_long(const void *pa, const void *pb)`
(b) `long long count_n_unique(long long *a, long long n)`
(c) `long long count_eq_classes_by_rows_sum(Matrix m)`

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include "libs/algorithms/matrix/matrix.h"

int cmp_long_long(const void *pa, const void *pb) {
    long long a = *(long long *) pa;
    long long b = *(long long *) pb;

    if (a < b)
        return -1;

    if (a > b)
        return 1;

    return 0;
}
```

```
long long count_n_unique(long long *a, long long n) {
    if (!n)
        return 0;

    qsort(a, n, sizeof(long long), cmp_long_long);

    long long count = 1;

    for (long long i = 1; i < n; i++)
        if (a[i] != a[i - 1])
            count++;

    return count;
}


long long count_eq_classes_by_rows_sum(Matrix m) {
    if (!m.n_rows)
        return 0;

    long long *row_sums = (long long *) malloc(m.n_rows *
sizeof(long long));

    for (long long i = 0; i < m.n_rows; i++) {
        row_sums[i] = 0;

        for (long long j = 0; j < m.n_cols; j++)
            row_sums[i] += m.values[i][j];
    }

    long long n_unique = count_n_unique(row_sums, m.n_rows);

    free(row_sums);

    return n_unique;
}
```

```c
int main(void) {
    printf("Enter the size of the square matrix\n");

    printf("N (rows): ");
    long long n;
    scanf("%lld", &n);

    printf("M (cols): ");
    long long m;
    scanf("%lld", &m);

    printf("Enter the elements of the matrix size of
%lldx%lld\n", n, m);

    Matrix matrix = get_matrix(n, m);
    input_matrix(matrix);

    long long n_unique = count_eq_classes_by_rows_sum(matrix);
    printf("Result: %lld", n_unique);

    free_matrix(matrix);

    return 0;
}
```

Задача №13.

Дан массив матриц одного размера. Определить число матриц, строки которых упорядочены по неубыванию элементов.

(a) `bool is_non_descending_sorted(long long *a, long long n)`
(b) `bool has_all_non_descending_rows(Matrix m)`
(c) `long long count_non_descending_rows_matrices(Matrix *ms, long long n_matrix)`

Код программы:

```c
#include <stdio.h>
#include <stdbool.h>
#include "libs/algorithms/matrix/matrix.h"

bool is_non_descending_sorted(long long *a, long long n) {
    for (long long i = 1; i < n; i++)
        if (a[i] < a[i - 1])
            return false;

    return true;
}

bool has_all_non_descending_rows(Matrix m) {
    for (long long i = 0; i < m.n_rows; i++)
        if (!is non descending sorted(m.values[i], m.n cols))
            return false;

    return true;
}

long long count non descending rows matrices(Matrix *ms, long
long n_matrix) {
    long long count = 0;

    for (long long i = 0; i < n_matrix; i++)
        if (has_all_non_descending_rows(ms[i]))
            count++;

    return count;
}
```

```
int main(void) {

    printf("Enter the size of array of matricies: ");
    long long size;
    scanf("%lld", &size);

    printf("Enter the size of matrices\n");

    printf("N (rows): ");
    long long n;
    scanf("%lld", &n);

    printf("M (cols): ");
    long long m;
    scanf("%lld", &m);

    printf("Enter %lld matrices of the size %lldx%lld\n", size,
n, m);

    Matrix *array_of_matrices = get_array_of_matrix(size, n, m);
    input array of matrix(array of matrices, size);

    long long count =
count_non_descending_rows_matrices(array_of_matrices, size);
    printf("Result: %lld", count);

    free_array_of_matrix(array_of_matrices, size);

    return 0;
}
```

Вывод: в ходе выполнения работы были получены навыки работы с многомерными массивами.