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 №15

По дисциплине: «Основы программирования»

Тема: «Создание библиотеки для работы с многомерными массивами»

Выполнил: студент группы ВТ-231

Борченко Александр Сергеевич

Проверили:

Черников Сергей Викторович

Новожен Никита Викторович

Белгород 2024

Цель работы: закрепление навыков создания библиотек, структур; получение навыков работы с многомерными массивами.

Содержание работы:

Заголовочный файл matrix.h:

```c
#include <stdio.h>
#include <malloc.h>
#include <stdint.h>
#include <stdbool.h>
#include <assert.h>
#include <memory.h>
typedef struct matrix {
    int **values; // элементы матрицы
    int nRows; // количество рядов
    int nCols; // количество столбцов
} matrix;

typedef struct position {
    int rowIndex;
    int colIndex;
} position;
```

Файл реализации matrix.c:

```c
#include "matrix.h"

//(критерий) вычисление суммы для одномерного массива
int getSum(int *a, int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += a[i];
    }
    return sum;
}

//размещает в динамической памяти матрицу размером nRows на nCols. Возвращает
матрицу.
matrix getMemMatrix(int nRows, int nCols) {
    int **values = (int **) malloc(sizeof(int*) * nRows);
    for (int i = 0; i < nRows; i++)
        values[i] = (int *) malloc(sizeof(int) * nCols);
    return (matrix){values, nRows, nCols};
}
/*размещает в динамической памяти массив из nMatrices матриц размером nRows
на nCols.
Возвращает указатель на нулевую матрицу */
matrix *getMemArrayOfMatrices(int nMatrices, int nRows, int nCols) {
    matrix *ms = (matrix*) malloc(sizeof(matrix) * nMatrices);
    for (int i = 0; i < nMatrices; i++)
        ms[i] = getMemMatrix(nRows, nCols);
    return ms;
}
//освобождает память, выделенную под хранение матрицы m
void freeMemMatrix(matrix *m) {
    for (size_t i = 0; i < m->nRows; i++) {
        free(m->values[i]);
    }

    free(m->values);

    m->values = NULL;
    m->nCols = 0;
    m->nRows = 0;
}
//освобождает память, выделенную под хранение массива ms из nMatrices матриц.
void freeMemMatrices(matrix *ms, int nMatrices) {
    for (size_t i = 0; i < nMatrices; i++)
```

```
        freeMemMatrix(&ms[i]);
}
//ввод матрицы m.
void inputMatrix(matrix *m) {
    for (size_t i = 0; i < m->nRows; i++)
        for (size_t j = 0; j < m->nCols; j++)
            scanf("%d", &m->values[i] [j]);
}
// ввод массива из nMatrices матриц, хранящейся по адресу ms.
void inputMatrices(matrix *ms, int nMatrices) {
    for (size_t i = 0; i < nMatrices; i++)
        inputMatrix(&ms[i]);
}
//вывод матрицы m.
void outputMatrix(matrix m) {
    for (size_t i = 0; i < m.nRows; i++)
        for (size_t j = 0; j < m.nCols; j++)
            printf("%d", m.values[i] [j]);
}
//вывод массива из nMatrices матриц, хранящейся по адресу ms.
void outputMatrices(matrix *ms, int nMatrices) {
    for (size_t i = 0; i < nMatrices; i++)
        outputMatrix(ms[i]);
}
//обмен строк с порядковыми номерами i1 и i2 в матрице m.
void swapRows(matrix m, int i1, int i2) {
    int *imatrix = m.values[i1];

    memcpy(&m.values[i1], &m.values[i2], sizeof(int*));
    memcpy(&m.values[i2], &imatrix, sizeof(int*));
}
//обмен колонок с порядковыми номерами j1 и j2 в матрице m.
void swapColumns(matrix *m, int j1, int j2) {
    assert(j1 < m->nCols || j2 < m->nCols);

    for (int i = 0; i < m->nRows; ++i) {
        int temp = m->values[i][j1];
        m->values[i][j1] = m->values[i][j2];
        m->values[i][j2] = temp;
    }
}

/* выполняет сортировку вставками строк
матрицы m по неубыванию значения функции criteria применяемой для
строк. */
void insertionSortRowsMatrixByRowCriteria(matrix m, int (*criteria)(int*,
int)) {
    int *values = malloc(sizeof(int) * m.nRows);

    for (size_t i = 0; i < m.nRows; i++)
        values[i] = criteria(m.values[i], m.nCols);

    for (size_t i = 1; i < m.nRows; i++) {

        int j = i - 1;

        while (values[i] < values[j] && j >= 0) {
            values[j + 1] = values[j];
            swapRows(m, j + 1, j);
            --j;
        }

        values[j + 1] = values[i];
```

```
    }
}
/*выполняет сортировку выбором столбцов
матрицы m по неубыванию значения функции criteria применяемой для столбцов
*/
void selectionSortColsMatrixByColCriteria(matrix *m, int (*criteria)(int *,
int)) {
    int temp[m->nCols];
    for (int i = 0; i < m->nCols; ++i) {
        int temp_column[m->nRows];
        for (int j = 0; j < m->nRows; ++j)
            temp_column[j] = m->values[j][i];


        int result = criteria(temp_column, m->nCols);
        temp[i] = result;
    }

    int min_pos, temp_pos;
    for (int i = 0; i < m->nCols; i++) {
        min_pos = i;
        for (int j = i + 1; j < m->nCols; j++)
            if (temp[min_pos] > temp[j])
                min_pos = j;
        temp_pos = temp[min_pos];
        temp[min_pos] = temp[i];
        temp[i] = temp_pos;

        swapColumns(m, min_pos, i);
    }
}

/*возвращает 'истина', если матрица m
является квадратной, ложь – в противном случае */
bool isSquareMatrix(matrix *m) {
    return m->nRows == m->nCols;
}
//возвращает 'истина', если матрицы m1 и m2 равны, иначе - ложь
bool areTwoMatricesEqual(matrix *m1, matrix *m2) {
    if (m1->nRows != m2->nRows || m1->nCols != m2->nCols)
        return 0;
    for (int i = 0; i < m1->nRows; ++i) {
        if (memcmp(m1->values[i], m2->values[i], sizeof(int) * m2->nCols) !=
0)
            return 0;
    }
    return 1;
}

//возвращает 'истина', если матрица m является единичной, иначе - ложь
bool isEMatrix(matrix *m) {
//т.к E матрица квадратная, то проверяем:
    if (!isSquareMatrix(m))
        return 0;

    for (size_t i = 0; i < m->nRows; i++) {
        if (m->values[i] [i] != 1)
            return 0;

        for (size_t j = 0; j < m->nCols; j++) {
            if (i != j && m->values[i] [j] != 0)
                return 0;
        }
```

```
    }

    return 1;
}
//возвращает 'истина', если матрица m является симметричной, иначе - ложь
bool isSymmetricMatrix(matrix *m) {
    if (!isSquareMatrix(m))
        return 0;

    for (size_t i = 0; i < m->nRows; i++) {
        for ( size_t j = 0; j < m->nCols; j++) {
            if ( m->values[i] [j] != m->values[j] [i])
                return 0;
        }
    }

    return 1;
}
//транспонирует квадратную матрицу m
//через стандартный swap не получилось, переделаю:
void transposeSquareMatrix(matrix *m) {
    if (!isSquareMatrix(m))
        return;

    matrix temp = getMemMatrix(m->nRows, m->nCols);

    for (int i = 0; i < m->nRows; i++) {
        for (int j = 0; j < m->nCols; j++) {
            temp.values[j][i] = m->values[i][j];
        }
    }

    m->values = temp.values;
}

//транспонирует матрицу m
void transposeMatrix(matrix *m) {
    matrix result = getMemMatrix(m->nCols, m->nRows);

    for (int col = 0; col < m->nCols; col++) {
        for (int row = 0, j = col; row < m->nRows; row++) {
            result.values[col] [row] = m->values[row] [j];
        }
    }

    memcpy(m, &result, sizeof(matrix));
}
//возвращает позицию минимального элемента матрицы m
position getMinValuePos(matrix m) {
    position pos;

    int min = INT_MAX;

    for (size_t i = 0; i < m.nRows; i++) {
        for (size_t j = 0; j < m.nCols; j++) {
            if (m.values[i] [j] < min) {
                min = m.values[i] [j];
                pos.rowIndex = i + 1;
                pos.colIndex = j + 1;
            }
        }
    }
```

```
    return pos;
}
//возвращает позицию максимального элемента матрицы m
position getMaxValuePos(matrix m) {
    position pos;

    int max = INT_MIN;

    for (size_t i = 0; i < m.nRows; i++) {
        for (size_t j = 0; j < m.nCols; j++) {
            if (m.values[i][j] > max) {
                max = m.values[i][j];
                pos.rowIndex = i + 1;
                pos.colIndex = j + 1;
            }
        }
    }

    return pos;
}
/*возвращает матрицу размера nRows на nCols, построенную из элементов массива
a
 * функция из методички */
matrix createMatrixFromArray(const int *a, int nRows, int nCols) {
    matrix m = getMemMatrix(nRows, nCols);
    int k = 0;
    for (int i = 0; i < nRows; i++)
        for (int j = 0; j < nCols; j++)
            m.values[i][j] = a[k++];
    return m;
}
/*возвращает указатель на нулевую матрицу массива из nMatrices матриц,
размещенных
в динамической памяти, построенных из элементов массива a.
Функция также из методички */
matrix *createArrayOfMatrixFromArray(const int *values,
                                     size_t nMatrices, size_t nRows, size_t
nCols) {

    matrix *ms = getMemArrayOfMatrices(nMatrices, nRows, nCols);

    int l = 0;
    for (size_t k = 0; k < nMatrices; k++)
        for (size_t i = 0; i < nRows; i++)
            for (size_t j = 0; j < nCols; j++)
                ms[k].values[i][j] = values[l++];
    return ms;
}
//подсчет количества value значений
int countValues(const int *a, int n, int value) {
    int count = 0;
    for (size_t i = 0; i < n; i++) {
        if (a[i] == value) {
            ++count;
        }
    }

    return count;
}
//возвращает количество строк в которых все элементы равны 0.
int countZeroRows(matrix m) {
    int result = 0;
    for (size_t i = 0; i < m.nRows; i++) {
```

```
        int count = countValues(m.values[i], m.nCols, 0);
//Если количество найденных нулевых элементов = общему количеству столбцов в
матрице, то увеличивается result
        if (count == m.nCols)
            ++result;
    }

    return result;
}
}
```

Тесты:

```
//Тесты функций
void test_countZeroRows() {
    matrix m = createMatrixFromArray(
            (int[]) {
                    1, 1, 0,
                    0, 0, 0,
                    0, 0, 1,
                    0, 0, 0,
                    0, 1, 1,
            },
            5, 3
    );

    assert(countZeroRows(m) == 2);
    freeMemMatrix(&m);
}

void test_getMemMatrix() {
    matrix m = getMemMatrix(2, 2);
    assert(m.nCols == 2 && m.nRows == 2);
    // проверка выделения памяти
    assert(m.values != NULL);
    freeMemMatrix(&m);

    // проверка на других входных данных
    m = getMemMatrix(2, 9);
    assert(m.nCols == 9 && m.nRows == 2);
    assert(m.values != NULL);

    //освобождаем память
    freeMemMatrix(&m);
}

void test_getMemArrayOfMatrices() {
    matrix *ms = getMemArrayOfMatrices(2, 2, 2);
    // проверка на данные о колоннах и столбцах
    assert(ms->nCols == 2 && ms->nRows == 2);
    //проверка выделения памяти
    assert(ms != NULL);
    freeMemMatrices(ms, 2);

    ms = getMemArrayOfMatrices(9, 5, 2);
    // проверка на данные о колоннах и столбцах
    assert(ms->nCols == 2 && ms->nRows == 5);
    // проверка выделения памяти
    assert(ms != NULL);
    freeMemMatrices(ms, 2);
}

void test_freeMemMatrix() {
    // создаем и чистим массив
    matrix m = getMemMatrix(2, 2);
    freeMemMatrix(&m);
    assert(m.values == NULL);

    //проверка на других данных
    m = getMemMatrix(3, 6);
    freeMemMatrix(&m);
    assert(m.values == NULL);
}
```

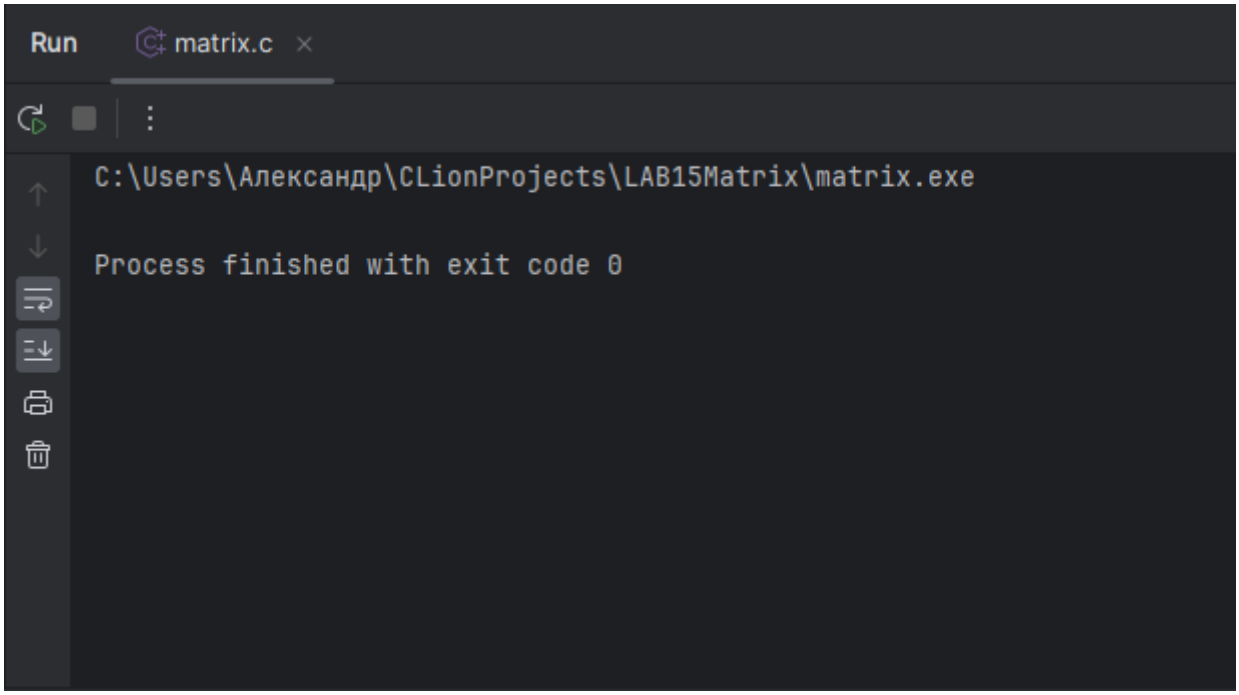
```
void test_swapRows() {
    // исходная матрица
    matrix m = createMatrixFromArray((int[])
            {4, 5, 6,
             1, 2, 3,},
             2, 3);
    // проверочная матрица
    matrix exp_res = createMatrixFromArray((int[]) {
        1, 2, 3,
        4, 5, 6},
                                           2, 3);
    swapRows(m, 0, 1);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_swapColumns() {
    // исходная матрица
    matrix m = createMatrixFromArray((int[]) {
        4, 5, 6,
        1, 2, 3,},
                                     2, 3);
    // проверочная матрица
    matrix exp_res = createMatrixFromArray((int[]) {
        5, 4, 6,
        2, 1, 3,},
        2, 3);

    swapColumns(&m, 0, 1);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_insertionSortRowsMatrixByRowCriteria() {
    //исходная матрица
    matrix m = createMatrixFromArray((int[]) {
        3, 3, 3,
        1, 1, 1,},
                                     2, 3);
    //проверочная матрица
    matrix exp_res = createMatrixFromArray((int[]) {
        1, 1, 1,
        3, 3, 3,},
                                           2, 3);

    insertionSortRowsMatrixByRowCriteria(m, getSum);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_selectionSortColsMatrixByColCriteria() {
    // исходная матрица
    matrix m = createMatrixFromArray((int[]) {
        1, 3, 1,
        1, 3, 1,},
                                     2, 3);
    // проверочная матрица
```

```
    matrix exp_res = createMatrixFromArray((int[]) {
        1, 1, 3,
        1, 1, 3,},
                                            2, 3);

    selectionSortColsMatrixByColCriteria(&m, getSum);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_isSquareMatrix() {
    //проверка квадратной матрицы
    matrix m = getMemMatrix(4, 4);
    assert(isSquareMatrix(&m));
    freeMemMatrix(&m);

    // проверка с не квадратной матрицей
    m = getMemMatrix(5, 8);
    assert(!isSquareMatrix(&m));
    freeMemMatrix(&m);
}

void test_areTwoMatricesEqual() {
    matrix m = createMatrixFromArray((int[]) {
        5, 4, 3,

        2, 1, 0,},
                                        2, 3);

    matrix exp_res = createMatrixFromArray((int[]) {
        5, 4, 3,
        2, 1, 0,},
                                            2, 3);

    //матрицы равны
    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);

    m = createMatrixFromArray((int[]) {
        1, 3, 1,
        1, 3, 3,},
                                2, 3);

    exp_res = createMatrixFromArray((int[]) {
        8, 6, 4,
        2, 3, 3,},
                                        2, 3);

    //матрицы не равны
    assert(!areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_isEMatrix() {
    matrix m = createMatrixFromArray((int[]) {
        1, 0, 0,
        0, 1, 0,
        0, 0, 1},
                                        3, 3);
```

```
    assert(isEMatrix(&m));

    matrix m_1 = createMatrixFromArray((int[]) {
        0, 1, 0,
        0, 1, 0,
        0, 1, 0},
                                        3, 3);
    assert(isEMatrix(&m_1) == 0);
}

void test_isSymmetricMatrix() {
    matrix m = createMatrixFromArray((int[]) {
        1, 0, 0,
        0, 1, 0,
        0, 0, 1},
                                      3, 3);
    assert(isSymmetricMatrix(&m));

    matrix m_1 = createMatrixFromArray((int[]) {
        1, 0, 0,
        0, 1, 0},
                                       2, 3);
    assert(isSymmetricMatrix(&m_1) == 0);
}

void test_transposeSquareMatrix() {
    matrix m = createMatrixFromArray((int[]) {
        1, 2, 3,
        4, 5, 0,
        -1, 3, 4},
                                      3, 3);
    matrix exp_res = createMatrixFromArray((int[]) {
        1, 4, -1,
        2, 5, 3,
        3, 0, 4},
                                           3, 3);

    transposeSquareMatrix(&m);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_transposeMatrix() {
    matrix m = createMatrixFromArray((int[]) {
        1, 2, 3,
        4, 5, 6,},
                                      2, 3);
    matrix exp_res = createMatrixFromArray((int[]) {
        1, 4,2,
        5,3, 6,},
                                           3, 2);

    transposeMatrix(&m);

    assert(areTwoMatricesEqual(&m, &exp_res));
    freeMemMatrix(&m);
    freeMemMatrix(&exp_res);
}

void test_getMinValuePos() {
    //нахождение минимума
```

```
    matrix m = createMatrixFromArray((int[]) {
        -1, 6, 3,
        18, 7, 0,},
                                2, 3);
    position p = getMinValuePos(m);

    assert(p.rowIndex == 1 && p.colIndex == 1);
    freeMemMatrix(&m);
}

void test_getMaxValuePos() {
    //нахождение максимума
    matrix m = createMatrixFromArray((int[]) {
        11, 10, 10,
        4, 15, 12,},
                                2, 3);
    position p1 = getMaxValuePos(m);

    assert(p1.rowIndex == 2 && p1.colIndex == 2);
    freeMemMatrix(&m);
}


void test() {
    //тесты взаимодействия с памятью
    test_getMemMatrix();
    test_getMemArrayOfMatrices();
    test_freeMemMatrix();

    //тест перераспределяющих функций
    test_swapRows();
    test_swapColumns();

    //тест для упорядочивания строк и столбцов
    test_insertionSortRowsMatrixByRowCriteria();
    test_selectionSortColsMatrixByColCriteria();

    //тест с методички
    test_countZeroRows();

    //тесты функций-предикаты
    test_isSquareMatrix();
    test_areTwoMatricesEqual();
    test_isEMatrix();
    test_isSymmetricMatrix();

    //тесты функций преобразования матриц
    test_transposeSquareMatrix();
    test_transposeMatrix();

    //тест функции для поиска минимального и максимального элемента матрицы
    test_getMinValuePos();
    test_getMaxValuePos();
}
```

Результат тестирования:



Результат выполнения:

```
Александр@DESKTOP-8QVSDRR MINGW64 ~/CLionProjects/LAB15Matrix (master)
$ git log --stat -- matrix.c
commit 0733eae0f937a34baa800f7b320eb563402f4ca8 (HEAD -> master, origin/master)
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Tue Mar 12 19:12:16 2024 +0300

    bug fix / complete test

 matrix.c | 238 ++++++++++++++++++++++++++++++++++++++++++++++++++++++--------
 1 file changed, 210 insertions(+), 28 deletions(-)

commit c259b9eb3984e48c430fdab66527f984f9741122
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Mon Mar 11 18:52:46 2024 +0300

    Исправлены функции и написана часть тестов

 matrix.c | 184 ++++++++++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 137 insertions(+), 47 deletions(-)

commit 12555686d71307fdc343e8a8fa74bc0d7d43a8eb
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Mon Mar 11 17:20:02 2024 +0300

    bug fix func freememmatrix
:...skipping...
commit 0733eae0f937a34baa800f7b320eb563402f4ca8 (HEAD -> master, origin/master)
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Tue Mar 12 19:12:16 2024 +0300

    bug fix / complete test

 matrix.c | 238 ++++++++++++++++++++++++++++++++++++++++++++++++++++++--------
 1 file changed, 210 insertions(+), 28 deletions(-)

commit c259b9eb3984e48c430fdab66527f984f9741122
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Mon Mar 11 18:52:46 2024 +0300

    Исправлены функции и написана часть тестов

 matrix.c | 184 ++++++++++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 137 insertions(+), 47 deletions(-)

commit 12555686d71307fdc343e8a8fa74bc0d7d43a8eb
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Mon Mar 11 17:20:02 2024 +0300

    bug fix func freememmatrix

 matrix.c | 27 ++++++++++++++++++++++++---
 1 file changed, 23 insertions(+), 4 deletions(-)

commit 0817fcb6052b37b0d04250a5d51836dd6ab30da7
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Sun Mar 10 10:56:41 2024 +0300

    refactoring / test / bug fix

 matrix.c | 85 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-----
 1 file changed, 79 insertions(+), 6 deletions(-)

commit d4baf29863c7da77551e06f46d3f06da7fba84df
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Fri Mar 8 20:35:54 2024 +0300

    Сделаны функции из пункта 7-8, проверены старые функции

 matrix.c | 59 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 59 insertions(+)
```

```
 matrix.c | 27 +++++++++++++++++++++++----
 1 file changed, 23 insertions(+), 4 deletions(-)

commit 0817fcb6052b37b0d04250a5d51836dd6ab30da7
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Sun Mar 10 10:56:41 2024 +0300

    refactoring / test / bug fix

 matrix.c | 85 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-----
 1 file changed, 79 insertions(+), 6 deletions(-)

commit d4baf29863c7da77551e06f46d3f06da7fba84df
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Fri Mar 8 20:35:54 2024 +0300

    Сделаны функции из пункта 7-8, проверены старые функции

 matrix.c | 59 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 59 insertions(+)

commit e589ae1b87c234e5d5a7bc6e4aae1bcc65e77fa7
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Fri Mar 8 18:47:49 2024 +0300

    Исправлен 4 пункт, сделан 5-6

 matrix.c | 114 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
 1 file changed, 113 insertions(+), 1 deletion(-)

commit 03fcdfc6f1bf8c7ff8d6a67f02bd057b01549491
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Fri Mar 8 13:25:28 2024 +0300

    Сделал пункты 3-4

 matrix.c | 46 ++++++++++++++++++++++++++++++++++++++++++++--
 1 file changed, 44 insertions(+), 2 deletions(-)

commit 759003821f9a56bab78200885a2369e630a00acc
Author: Александр <alexanders.borchenko@gmail.com>
Date:   Wed Mar 6 20:51:20 2024 +0300

    Реализована функция freeMemMatrix

 matrix.c | 26 ++++++++++++++++++++++++++
 1 file changed, 26 insertions(+)
:
```

Вывод: в ходе выполнения лабораторной работы я закрепил навыки создания библиотек, структур, написал библиотеку matrix и получил навыки работы с многомерными массивами.