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»

по теме: «Создание библиотеки для работы с многомерными массивами»

Выполнил/a: ст. группы ВТ-231

Петрухин Яков Ильич

Проверили:

Черников Сергей Викторович

Новожен Никита Викторович

Белгород, 2024 г.

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

Код библиотеки:

#include <malloc.h>

#include <stdbool.h>

#include <string.h>

#include <stdio.h>

#include "matrix.h"

matrix getMemMatrix (int nRows, int nCols)

{

    int \*\*values = (int \*\*) malloc(sizeof(int\*) \* (unsigned int)nRows);

    for (int i = 0; i < nRows; i++)

    {

        values[i] = (int \*) malloc(sizeof(int) \* (unsigned int)nCols);

    }

    return (matrix){values, nRows, nCols};

}

void freeMemMatrix (matrix \*m)

{

    for (int i = 0; i < m->nRows; i++)

    {

        free(m->values[i]);

    }

    free(m->values);

}

matrix \*getMemArrayOfMatrices(int nMatrices, int nRows, int nCols)

{

    matrix \*ms = (matrix\*) malloc(sizeof(matrix) \* (unsigned int)nMatrices);

    for (int i = 0; i < nMatrices; i++)

    {

        ms[i] = getMemMatrix(nRows, nCols);

    }

    return ms;

}

void freeMemMatrices(matrix \*ms, int nMatrices)

{

    for (int i = 0; i < nMatrices; i++)

    {

        freeMemMatrix(&ms[i]);

    }

    free(ms);

}

void inputMatrix(matrix \*m)

{

    int \*row;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            scanf("%d", &row[j]);

        }

    }

}

void outputMatrix(matrix \*m)

{

    int \*row;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            printf("%d ", row[j]);

        }

        printf("\n");

    }

}

void inputMatrices(matrix \*ms, int nMatrices)

{

    for(int i = 0; i < nMatrices; i++)

    {

        inputMatrix(&ms[i]);

    }

}

void outputMatrices(matrix \*ms, int nMatrices)

{

    for(int i = 0; i < nMatrices; i++)

    {

        outputMatrix(&ms[i]);

    }

}

void swapRows(matrix \*m, int i1, int i2)

{

    int \*row;

    row = m->values[i1];

    m->values[i1] = m->values[i2];

    m->values[i2] = row;

}

void swapColumns(matrix \*m, int j1, int j2)

{

    int \*row;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            int num\_mem;

            num\_mem = row[j1];

            row[j1] = row[j2];

            row[j2] = num\_mem;

        }

    }

}

void insertionSortRowsMatrixByRowCriteria (matrix \*m, int (\*criteria)(int\*, int))

{

    int \*sum;

    sum = malloc(sizeof(int) \* (unsigned int)m->nRows);

    for(int i = 0; i < m->nRows; i++) // получаем временный массив критериев

    {

        sum[i] = criteria(m->values[i], m->nCols);

    }

    // сортировка вставками

    for (int i = 1; i < m->nRows; i++)

    {

        int newElement = sum[i];

        int \*new\_row = m->values[i];

        int location = i - 1;

        while(location >= 0 && sum[location] > newElement)

        {

            sum[location + 1] = sum[location];

            location = location - 1;

            swapRows(m, location + 1, location);

        }

        sum[location + 1] = newElement;

        m->values[location + 1] = new\_row;

    }

    free(sum);

}

int getSum(int \*a, int n)

{

    int sum = 0;

    for(int i = 0; i < n; i++)

    {

        sum += a[i];

    }

    return sum;

}

void selectionSortColsMatrixByColCriteria (matrix \*m, int (\*criteria)(int\*, int))

{

    int \*arr, \*sum;

    arr = malloc(sizeof(int) \* (unsigned int)m->nRows);

    sum = malloc(sizeof(int) \* (unsigned int)m->nCols);

    for (int i = 0; i < m->nCols; i++)

    {

        getMatrixColumn (m, i, arr);

        sum[i] = criteria(arr, m->nRows);

    }

    // сортировка выбором

    for (int i = 0; i < m->nCols; i++)

    {

        int minPosition = i;

        for (int j = i + 1; j < m->nCols; j++)

        {

            if (sum[minPosition] > sum[j])

            {

                minPosition = j;

            }

        }

        int tmp = sum[minPosition];

        sum[minPosition] = sum[i];

        sum[i] = tmp;

        swapColumns (m, minPosition, i);

    }

    free(sum);

    free(arr);

}

void getMatrixColumn (matrix \*m, int num, int \*column)

{

    for(int i = 0; i < m->nRows; i++)

    {

        int \*row = m->values[i];

        column[i] = row[num];

    }

}

bool isSquareMatrix(matrix \*m)

{

    return m->nCols == m->nRows;

}

bool areTwoMatricesEqual(matrix \*m1, matrix \*m2)

{

    bool answ = true;

    int \*row1, \*row2;

    if (!isSquareMatrix(m1) || !isSquareMatrix(m2))

    {

        answ = false;

    }

    if(answ != false)

    {

        for(int i = 0; i < m1->nRows; i++)

        {

            row1 = m1->values[i];

            row2 = m2->values[i];

            if(memcmp(row1, row2, (unsigned int)m1->nCols) != 0)

            {

                answ = false;

                break;

            }

        }

    }

    return answ;

}

bool isEMatrix(matrix \*m)

{

    int \*row;

    bool answ = true;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            if(j == i)

            {

                if(row[j] != 1)

                {

                    answ = false;

                    break;

                }

            }

            else

            {

                if(row[j] != 0)

                {

                    answ = false;

                    break;

                }

            }

        }

        if(!answ)

        {

            break;

        }

    }

    return answ;

}

bool isSymmetricMatrix(matrix \*m)

{

    int \*row;

    bool answ = true;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = i + 1; j < m->nCols; j++)

        {

            if(row[j] != getElementMatrix(m, j, i))

            {

                answ = false;

                break;

            }

        }

        if(!answ)

        {

            break;

        }

    }

    return answ;

}

int getElementMatrix(matrix \*m, int i, int j)

{

    int \*row = m->values[i];

    return row[j];

}

void putElementMatrix(matrix \*m, int i, int j, int val)

{

    int \*row = m->values[i];

    row[j] = val;

}

void transposeSquareMatrix(matrix \*m)

{

    if(isSquareMatrix(m))

    {

        int \*row;

        for(int i = 0; i < m->nRows; i++)

        {

            row = m->values[i];

            for(int j = i + 1; j < m->nCols; j++)

            {

                int temp = row[j];

                row[j] = getElementMatrix(m, j, i);

                putElementMatrix(m, j, i, temp);

            }

        }

    }

    else

    {

        transposeMatrix(m);

    }

}

void transposeMatrix(matrix \*m)

{

    if(isSquareMatrix(m))

    {

        transposeSquareMatrix(m);

    }

    else

    {

        int \*row;

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

        for(int i = 0; i < m->nRows; i++)

        {

            row = m->values[i];

            for(int j = 0; j < m->nCols; j++)

            {

                putElementMatrix(&new, j, i, row[j]);

            }

        }

        freeMemMatrix(m);

        m = &new;

    }

}

position getMinValuePos(matrix \*m)

{

    int \*row, min = INT16\_MAX;

    position pos;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            if(min > row[j])

            {

                min = row[j];

                pos.rowIndex = i;

                pos.colIndex = j;

            }

        }

    }

    return pos;

}

position getMaxValuePos(matrix \*m)

{

    int \*row, max = INT16\_MIN;

    position pos;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        for(int j = 0; j < m->nCols; j++)

        {

            if(max < row[j])

            {

                max = row[j];

                pos.rowIndex = i;

                pos.colIndex = j;

            }

        }

    }

    return pos;

}

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;

}

matrix \*createArrayOfMatrixFromArray(const int \*values, int nMatrices, int nRows, int nCols)

{

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

    int l = 0;

    for (int k = 0; k < nMatrices; k++)

    {

        for (int i = 0; i < nRows; i++)

        {

            for (int j = 0; j < nCols; j++)

            {

                ms[k].values[i][j] = values[l++];

            }

        }

    }

    return ms;

}

Тестовая программа:

#include "matrix.h"

#include <assert.h>

int countZeroRows(matrix \*m);

void test\_countZeroRows()

{

    const int arr[] =

    {

        1, 1, 0,

        0, 0, 0,

        0, 0, 1,

        0, 0, 0,

        0, 1, 1,

    };

    matrix m = createMatrixFromArray(arr, 5, 3);

    assert(countZeroRows(&m) == 2);

    outputMatrix(&m);

}

int countZeroRows(matrix \*m)

{

    int \*row, count = 0;

    bool answ;

    for(int i = 0; i < m->nRows; i++)

    {

        row = m->values[i];

        answ = true;

        for(int j = 0; j < m->nCols; j++)

        {

            if(row[j] != 0)

            {

                answ = false;

                break;

            }

        }

        if(answ)

        {

            count++;

        }

    }

    return count;

}

int main()

{

    test\_countZeroRows();

    return 0;

}

Консоль вывода тестовой программы:

\* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe

1 1 0

0 0 0

0 0 1

0 0 0

0 1 1

\* Terminal will be reused by tasks, press any key to close it.

Ссылка на открытый репозиторий:

<https://github.com/Frozzie/uni_homework_matrices>

Вывод: Я научился создавать и проверять библиотеку функций для работы с многомерным массивом.