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: ст.группы
Игнатьев Артур Олегович

Проверил:
Преподаватель Притчин Иван Сергеевич
Преподаватель Черников Сергей Викторович

Белгород 2023г.

Лабораторная работа «Создание библиотеки для работы с многомерными массивами»

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

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Решения задач:

  –Текст задания.

  – Исходный код (в том числе и тестов).

  – Задания со звездочкой не являются обязательными, но их решение требуется для получения максимального балла.
- Ссылка на открытый репозиторий с решением.
- Скриншот с историей коммитов.

Требования:

- Выполните автоматизированное тестирование разработанной библиотеки

Задания к лабораторной работе:

1. В заголовочном файле libs\data_structures\matrix\matrix.h объявите структуру ’матрица’ и ’позиция’:

```
typedef struct matrix {
    int **values; //элементы матрицы
    int nRows; //количество рядов
    int nCols; //количество столбцов
} matrix;

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

2. В библиотеке matrix реализуйте функции для размещения в динамической памяти матриц:

(a) matrix getMemMatrix(int nRows, int nCols) – размещает в динамической памяти матрицу размером nRows на nCols. Возвращает матрицу.

```
matrix getMemMatrix(int nRows, int nCols) {
    int **values = (int **) malloc(nRows * sizeof(int *));

    if (values == NULL) {
        fprintf(stderr, "bad alloc");
        exit(1);
    }

    for (int i = 0; i < nRows; i++) {
        values[i] = (int *) malloc(sizeof(int) * nCols);
        if (values[i] == NULL) {
            fprintf(stderr, "bad alloc");
            exit(1);
        }
    }
    return (matrix) {values, nRows, nCols};
}
```

(b) matrix *getMemArrayOfMatrices(int nMatrices, int nRows, int 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(nMatrices * sizeof(matrix));
    if (ms == NULL) {
        fprintf(stderr, "bad alloc");
        exit(1);
    }

    for (int i = 0; i < nMatrices; i++)
        ms[i] = getMemMatrix(nRows, nCols);

    return ms;
}
```

(c) void freeMemMatrix(matrix *m) – освобождает память, выделенную под хранение матрицы m.

```
void freeMemMatrix(matrix *m) {
    for (int i = 0; i < m->nRows; i++)
        free(m->values[i]);
    free(m->values);
    m->values = NULL;
    m->nRows = 0;
    m->nCols = 0;
}
```

(d) void freeMemMatrices(matrix *ms, int nMatrices) – освобождает память, выделенную под хранение массива ms из nMatrices матриц.

```
void freeMemMatrices(matrix *ms, int nMatrices) {
    for (int i = 0; i < nMatrices; i++)
        freeMemMatrix(&ms[i]);
    free(ms);
}
```

3. В библиотеке matrix реализуйте функции для ввода и вывода матриц:

(a) void inputMatrix(matrix *m) – ввод матрицы m.

```
void inputMatrix(matrix m) {
    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; cIndex++) {
            scanf("%d", &m.values[rIndex][cIndex]);
        }
    }
}
```

(b) void inputMatrices(matrix *ms, int nMatrices) – ввод массива из nMatrices матриц, хранящейся по адресу ms.

```
void inputMatrices(matrix *ms, int nMatrices) {
    for (int i = 0; i < nMatrices; i++) {
        inputMatrix(ms[i]);
    }
}
```

(c) void outputMatrix(matrix m) – вывод матрицы m.

```
void outputMatrix(matrix m) {
    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; cIndex++) {
            printf("%d ", m.values[rIndex][cIndex]);
        }
        printf("\n");
    }
}
```

(d) void outputMatrices(matrix *ms, int nMatrices) – вывод массива из nMatrices матриц, хранящейся по адресу ms.

```
void outputMatrices(matrix *ms, int nMatrices) {
    for (int i = 0; i < nMatrices; i++) {
        outputMatrix(ms[i]);
    }
}
```

4. В библиотеке matrix реализуйте функции для обмена строк и столбцов:

(a) void swapRows(matrix m, int i1, int i2) – обмен строк с порядковыми номерами i1 и i2 в матрице m.

```
void swapRows(matrix m, int i1, int i2) {
    assert(i1 < m.nRows && i2 < m.nRows);

    universalSwap(&m.values[i1], &m.values[i2], sizeof(int *));
}
```

(b) void swapColumns(matrix m, int j1, int j2) – 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 rIndex = 0; rIndex < m.nRows; rIndex++) {
        universalSwap(&m.values[rIndex][j1], &m.values[rIn-
dex][j2], sizeof(int));
    }
}
```

5. В библиотеке matrix реализуйте функции для упорядочивания строк и столбцов:

(a) void insertionSortRowsMatrixByRowCriteria(matrix m, int (*criteria)(int*, int)) – выполняет сортировку вставками строк матрицы m по неубыванию значения функции criteria применяемой для строк.

```
void insertionSortRowsMatrixByRowCriteria(matrix m, int (*crite-
ria)(int *, int)) {
    int *rows = (int *) malloc(sizeof(int) * m.nRows);

    for (int rIndex = 0; rIndex < m.nRows; ++rIndex)
        rows[rIndex] = criteria(m.values[rIndex], m.nCols);

    for (int rIndex = 1; rIndex < m.nRows; ++rIndex) {
        int curIndex = rIndex;
        while (curIndex > 0 && rows[curIndex] < rows[curIndex -
1]) {
            swapRows(m, curIndex, curIndex - 1);
            universalSwap(&rows[curIndex], &rows[curIndex - 1],
sizeof(int));
            curIndex--;
        }
    }
}
```

(b) void selectionSortColsMatrixByColCriteria(matrix m, int (*criteria)(int*, int)) – выполняет сортировку выбором столбцов матрицы m по неубыванию значения функции criteria применяемой для столбцов.

```
void insertionSortColsMatrixByColCriteria(matrix m, int (*crite-
ria)(int *, int)) {
    int *cols = (int *) malloc(sizeof(int) * m.nCols);
    int *curCols = malloc(sizeof(int) * m.nRows);

    for (int cIndex = 0; cIndex < m.nCols; ++cIndex) {
        for (int rIndex = 0; rIndex < m.nRows; ++rIndex) {
            curCols[rIndex] = m.values[rIndex][cIndex];
        }
        cols[cIndex] = criteria(curCols, m.nRows);
    }

    for (int cIndex = 1; cIndex < m.nCols; ++cIndex) {
        int curIndex = cIndex;
        while (curIndex > 0 && cols[curIndex] < cols[curIndex -
1]) {
            swapColumns(m, curIndex, curIndex - 1);
            universalSwap(&cols[curIndex], &cols[curIndex - 1],
sizeof(int));
            curIndex--;
        }
    }
}
```

6. В библиотеке matrix реализуйте следующие функции-предикаты:

- bool isSquareMatrix(matrix *m) – возвращает значение ’истина’, если матрица m является квадратной, ложь – в противном случае 34.

```
bool isSquareMatrix(matrix m) {
    return m.nRows == m.nCols;
}
```

- bool areTwoMatricesEqual(matrix *m1, matrix *m2) – возвращает значние ’истина’, если матрицы m1 и m2 равны, ложь – в противном случае.

```
bool areTwoMatricesEqual(matrix m1, matrix m2) {
    if (m1.nRows != m2.nRows || m1.nCols != m2.nCols)
        return false;

    for (int rIndex = 0; rIndex < m1.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m1.nCols; ++cIndex) {
            if (m1.values[rIndex][cIndex] != m2.values[rIn-
dex][cIndex])
                return false;
        }
    }
    return true;
}
```

- bool isEMatrix(matrix *m) – возвращает значение ’истина’, если матрица m является единичной, ложь – в противном случае.

```
bool isEMatrix(matrix m) {
    if (!isSquareMatrix(m))
        return false;

    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; ++cIndex) {
            if (rIndex == cIndex && m.values[rIndex][cIndex] !=
1
                || rIndex != cIndex && m.values[rIndex][cIndex]
!= 0)
                return false;
        }
    }
    return true;
}
```

- bool isSymmetricMatrix(matrix *m) – возвращает значение ’истина’, если матрица m является симметричной, ложь – в противном случае.

```
bool isSymmetricMatrix(matrix m) {
    if (!isSquareMatrix(m))
        return false;

    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; cIndex++) {
            int rowsIndex = cIndex;
            int colsIndex = rIndex;

            if (m.values[rIndex][cIndex] !=
m.values[rowsIndex][colsIndex])
                return false;
        }
    }
    return true;
}
```

7. В библиотеке matrix реализуйте следующие функции преобразования матриц:

- void transposeSquareMatrix(matrix *m) – транспонирует квадратную мтрицу m.

```
void transposeSquareMatrix(matrix m) {
    assert(isSquareMatrix(m));

    for (int rIndex = 0; rIndex < m.nRows; ++rIndex) {
        for (int cIndex = 0; cIndex < rIndex; ++cIndex) {
            if (rIndex != cIndex)
                universalSwap(&m.values[rIndex][cIndex],
&m.values[cIndex][rIndex], sizeof(int));
        }
    }
}
```

- void transposeMatrix(matrix *m) – транспонирует матрицу m.

```
void transposeMatrix(matrix *m) {
    matrix newMatrix = getMemMatrix(m->nCols, m->nRows);

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

    freeMemMatrix(m);
    *m = newMatrix;
}
```

8. В библиотеке matrix реализуйте функции для поиска минимального и максимального элемента матрицы:

- position getMinValuePos(matrix m) – возвращает позицию минимального элемента матрицы m.

```
position getMinValuePos(matrix m) {
    position minValuePos = {0, 0};

    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; cIndex++) {
            position currentPos = {rIndex, cIndex};
            if
(m.values[currentPos.rowIndex][currentPos.colIndex] <

m.values[minValuePos.rowIndex][minValuePos.colIndex])
                minValuePos = currentPos;
        }
    }
    return minValuePos;
}
```

- position getMaxValuePos(matrix m) – возвращает позицию максимального элемента матрицы m.

```
position getMaxValuePos(matrix m) {
    position maxValuePos = {0, 0};

    for (int rIndex = 0; rIndex < m.nRows; rIndex++) {
        for (int cIndex = 0; cIndex < m.nCols; cIndex++) {
            position currentPos = {rIndex, cIndex};
            if
(m.values[currentPos.rowIndex][currentPos.colIndex] >

m.values[maxValuePos.rowIndex][maxValuePos.colIndex])
                maxValuePos = currentPos;
        }
    }
    return maxValuePos;
}
```

9. Дополните библиотеку функциями для тестирования:

- matrix createMatrixFromArray(const int *a, size_t nRows, size_t nCols) – возвращает матрицу размера nRows на nCols, построенную из элементов массива a:

```
matrix createMatrixFromArray(const int *a, int nRows, int nCols)
{
    matrix m = getMemMatrix(nRows, nCols);

    int k = 0;
    for (int rIndex = 0; rIndex < nRows; rIndex++)
        for (int cIndex = 0; cIndex < nCols; cIndex++)
            m.values[rIndex][cIndex] = a[k++];

    return m;
}
```

- matrix *createArrayOfMatrixFromArray(const int *values, size_t nMatrices, size_t nRows, size_t nCols) – возвращает указатель на нулевую матрицу массива из nMatrices матриц, размещенных в динамической памяти, построенных из элементов массива a:

```
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 rIndex = 0; rIndex < nRows; rIndex++)
            for (int cIndex = 0; cIndex < nCols; cIndex++)
                ms[k].values[rIndex][cIndex] = values[l++];

    return ms;
}
```

Ссылка на репозиторий с библиотекой matrix:

https://github.com/NTK-Hub/Labs/tree/master/libs/data_structures/matrix