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 ПВ-223

Пахомов Владислав Андреевич

Проверили:

Притчин Иван Сергеевич

Черников Сергей Викторович

Код-ревьер: ст. группы ПВ-223

Голуцкий Георгий Юрьевич

Белгород 2023 г.

**Лабораторная работа № 15**

**Содержание отчёта:**

* Тема лабораторной работы.
* Цель лабораторной работы.
* Решения задач.
  + Текст задания.
  + Исходный код (в т.ч. и тестов).
* Ссылка на открытый репозиторий с решением.
* Скриншот с историей коммитов.
* Вывод по работе.

**Тема лабораторной работы:** Создание библиотеки для работы с многомерными массивами

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

**Решения задач:**

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;

1. В библиотеке matrix реализуйте функции для размещения в динамической памяти матриц:
   1. Matrix getMemMatrix(int nRows, int nCols) – размещает в динамиче- ской 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};  
}

* 1. Matrix \*getMemArrayOfMatrices(int nMatrices, int nRows, int nCols) – размещает в динамической памяти массив из матриц размером nRows на nCols. Возвращает указатель на нулевую матрицу.

Matrix \*getMemArrayOfMatrices(int nMatrices, int nRows, int nCols) {  
 Matrix \*matrixArray = (Matrix \*) malloc(sizeof(Matrix) \* nMatrices);  
 for (int i = 0; i < nMatrices; i++)  
 matrixArray[i] = getMemMatrix(nRows, nCols);  
 return matrixArray;  
}

* 1. 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);  
}

* 1. 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);  
}

1. В библиотеке matrix реализуйте функции для ввода и вывода матриц:
   1. void inputMatrix(Matrix m) - ввод матрицы m.

void inputMatrix(Matrix m) {  
 for (int i = 0; i < m.nRows; i++)  
 for (int j = 0; j < m.nCols; j++)  
 scanf("%d", &m.values[i][j]);  
}

* 1. 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]);  
}

* 1. void outputMatrix(Matrix m) - ввод матрицы m.

void outputMatrix(Matrix m) {  
 for (int i = 0; i < m.nRows; i++) {  
 for (int j = 0; j < m.nCols; j++)  
 printf("%d ", m.values[i][j]);  
  
 printf("\n");  
 }  
}

* 1. 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]);  
}

1. В библиотеке matrix реализуйте функции для обмена строк и столбцов:
   1. 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 && i1 >= 0 && i2 >= 0);  
  
 swap(m.values + i1, m.values + i2, sizeof(m.values[0]));  
}

void testSwapRows() {  
 Matrix m = createMatrixFromArray((int[]) {1, 2, 3, 4,  
 5, 6, 7, 8,  
 9, 10, 11, 12}, 3, 4);  
  
 swapRows(m, 0, 2);  
  
 Matrix expectedArray = createMatrixFromArray((int[]) {9, 10, 11, 12,  
 5, 6, 7, 8,  
 1, 2, 3, 4}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedArray);  
}

* 1. 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 && j1 >= 0 && j2 >= 0);  
  
 for (int i = 0; i < m.nRows; i++)  
 swap(&m.values[i][j1], &m.values[i][j2], sizeof(m.values[0][0]));  
}

void testSwapColumns() {  
 Matrix m = createMatrixFromArray((int[]) {1, 2, 3, 4,  
 5, 6, 7, 8,  
 9, 10, 11, 12}, 3, 4);  
  
 swapColumns(m, 0, 2);  
  
 Matrix expectedArray = createMatrixFromArray((int[]) {3, 2, 1, 4,  
 7, 6, 5, 8,  
 11, 10, 9, 12}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedArray);  
}

1. В библиотеке matrix реализуйте функции для упорядочивания строк и столбцов:
   1. 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 (\*criteria)(int \*, int)) {int \*keys = (int \*) malloc(sizeof(int) \* m.nRows);  
 for (int i = 0; i < m.nRows; i++)  
 keys[i] = criteria(m.values[i], m.nCols);  
  
 for (int i = 1; i < m.nRows; i++) {  
 int tKey = keys[i];  
 int\* tRow = m.values[i];  
 int j = i;  
 while (j > 0 && keys[j - 1] > tKey) {  
 keys[j] = keys[j - 1];  
 m.values[j] = m.values[j - 1];  
 j--;  
 }  
  
 keys[j] = tKey;  
 m.values[j] = tRow;  
 }  
  
 free(keys);  
}

int getSum(int \*array, int arraySize) {  
 int result = 0;  
  
 for (int i = 0; i < arraySize; i++)  
 result += array[i];  
  
 return result;  
}  
  
int getMin(int \*array, int arraySize) {  
 int result = arraySize == 0 ? 0 : array[0];  
  
 for (int i = 0; i < arraySize; i++)  
 if (array[i] < result)  
 result = array[i];  
  
 return result;  
}  
  
int getMax(int \*array, int arraySize) {  
 int result = arraySize == 0 ? 0 : array[0];  
  
 for (int i = 0; i < arraySize; i++)  
 if (array[i] > result)  
 result = array[i];  
  
 return result;  
}  
  
void testInsertionSortRowsMatrixByRowCriteria() {  
 Matrix m = createMatrixFromArray((int[]) {12, 6, 1, 4, *//23* 9, 10, 11, 5, *//35* 7, 2, 3, 8*/\*20\*/*}, 3, 4);  
  
 insertionSortRowsMatrixByRowCriteria(m, getSum);  
  
 Matrix expectedArray = createMatrixFromArray((int[]) {7, 2, 3, 8,  
 12, 6, 1, 4,  
 9, 10, 11, 5}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(expectedArray);  
  
 insertionSortRowsMatrixByRowCriteria(m, getMax);  
  
 expectedArray = createMatrixFromArray((int[]) {7, 2, 3, 8,  
 9, 10, 11, 5,  
 12, 6, 1, 4}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(expectedArray);  
  
 insertionSortRowsMatrixByRowCriteria(m, getMin);  
  
 expectedArray = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 7, 2, 3, 8,  
 9, 10, 11, 5}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedArray);  
}

* 1. void selectionSortColsMatrixByColCriteria(Matrix m, int (\*criteria)(int\*, int)) - выполняет сортировку выбором столбцов матрицы m по неубыванию значения функции criteria применяемой для столбцов

void selectionSortColsMatrixByColCriteria(Matrix m,  
 int (\*criteria)(int \*, int)) {  
 int \*keys = (int \*) malloc(sizeof(int) \* m.nCols);  
 for (int i = 0; i < m.nCols; i++) {  
 int \*column = (int \*) malloc(sizeof(int) \* m.nRows);  
  
 for (int j = 0; j < m.nRows; j++)  
 column[j] = m.values[j][i];  
  
 keys[i] = criteria(column, m.nRows);  
  
 free(column);  
 }  
  
 for (int i = 0; i < m.nCols - 1; i++) {  
 int minIndex = i;  
 for (int j = i + 1; j < m.nCols; j++)  
 if (keys[j] < keys[minIndex])  
 minIndex = j;  
  
 swapColumns(m, minIndex, i);  
 swap(keys + i, keys + minIndex, sizeof(keys[0]));  
 }  
  
 free(keys);  
}

void testSelectionSortColsMatrixByColCriteria() {Matrix m = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 selectionSortColsMatrixByColCriteria(m, getSum);  
  
 Matrix expectedArray = createMatrixFromArray((int[]) {1, 4, 6, 12,  
 11, 5, 10, 9,  
 3, 8, 2, 7}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(expectedArray);  
  
 selectionSortColsMatrixByColCriteria(m, getMax);  
  
 expectedArray = createMatrixFromArray((int[]) {4, 6, 1, 12,  
 5, 10, 11, 9,  
 8, 2, 3, 7}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(expectedArray);  
  
 selectionSortColsMatrixByColCriteria(m, getMin);  
  
 expectedArray = createMatrixFromArray((int[]) {1, 6, 4, 12,  
 11, 10, 5, 9,  
 3, 2, 8, 7}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(expectedArray, m));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedArray);  
}

1. В библиотеке matrix реализуйте следующие функции-предикаты:
   1. bool isSquareMatrix(Matrix m) - возвращает значение ’истина’, если матрица m является квадратной, ложь – в противном случае.

**bool** isSquareMatrix(Matrix m) {  
 return m.nRows == m.nCols;  
}

void testIsSquareMatrix() {  
 *// Why go through all this trouble for such a small function...* Matrix m = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(!isSquareMatrix(m));  
  
 freeMemMatrix(m);  
  
 m = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8,  
 1, 2, 3, 4}, 4, 4);  
  
 **assert**(isSquareMatrix(m));  
  
 freeMemMatrix(m);  
}

* 1. bool areTwoMatricesEqual(Matrix m1, Matrix m2) - возвращает значение ’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 i = 0; i < m1.nRows; i++)  
 if(memcmp(m1.values[i], m2.values[i], sizeof(int) \* m1.nCols))  
 return **false**;  
  
 return **true**;  
}

void testAreTwoMatricesEqual() {  
 *// Why go through all this trouble for such a small function...* Matrix m1 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
 Matrix m2 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(areTwoMatricesEqual(m1, m2));  
  
 freeMemMatrix(m1);  
 freeMemMatrix(m2);  
  
 m1 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 16, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
 m2 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(!areTwoMatricesEqual(m1, m2));  
  
 freeMemMatrix(m1);  
 freeMemMatrix(m2);  
  
 m1 = createMatrixFromArray((int[]) {12, 6,  
 9, 16  
 }, 2, 2);  
 m2 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(!areTwoMatricesEqual(m1, m2));  
  
 freeMemMatrix(m1);  
 freeMemMatrix(m2);  
}

* 1. 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 i = 0; i < m.nRows; i++)  
 for (int j = 0; j < m.nCols; j++)  
 if ((i == j) != m.values[i][j])  
 return **false**;  
  
 return **true**;  
}

void testIsEMatrix() {  
 Matrix m1 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(!isEMatrix(m1));  
  
 freeMemMatrix(m1);  
  
 m1 = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8,  
 1, 1, 1, 1}, 4, 4);  
  
 **assert**(!isEMatrix(m1));  
  
 freeMemMatrix(m1);  
  
 m1 = createMatrixFromArray((int[]) {1, 0, 0, 0,  
 0, 1, 0, 0,  
 0, 0, 1, 0,  
 0, 0, 0, 1}, 4, 4);  
  
 **assert**(isEMatrix(m1));  
  
 freeMemMatrix(m1);  
  
 m1 = createMatrixFromArray((int[]) {1, 0, 0, 0,  
 0, 1, 0, 1,  
 0, 0, 1, 0,  
 0, 0, 0, 1}, 4, 4);  
  
 **assert**(!isEMatrix(m1));  
  
 freeMemMatrix(m1);  
}

* 1. 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 i = 0; i < m.nRows; i++)  
 for (int j = i + 1; j < m.nCols; j++)  
 if (m.values[i][j] != m.values[j][i])  
 return **false**;  
  
 return **true**;  
}

void testIsSymmetricMatrix() {  
 Matrix m = createMatrixFromArray((int[]) {12, 6, 1, 4,  
 9, 10, 11, 5,  
 7, 2, 3, 8}, 3, 4);  
  
 **assert**(!isSymmetricMatrix(m));  
  
 freeMemMatrix(m);  
  
 m = createMatrixFromArray((int[]) {1, 3, 0,  
 3, 2, 6,  
 0, 6, 5}, 3, 3);  
  
 **assert**(isSymmetricMatrix(m));  
  
 freeMemMatrix(m);  
  
 m = createMatrixFromArray((int[]) {1, 0, 0,  
 0, 1, 0,  
 0, 0, 1}, 3, 3);  
  
 **assert**(isSymmetricMatrix(m));  
  
 freeMemMatrix(m);  
  
 m = createMatrixFromArray((int[]) {1, 5,  
 5, 7}, 2, 2);  
  
 **assert**(isSymmetricMatrix(m));  
  
 freeMemMatrix(m);  
  
 m = createMatrixFromArray((int[]) {2}, 1, 1);  
  
 **assert**(isSymmetricMatrix(m));  
  
 freeMemMatrix(m);  
}

1. В библиотеке matrix реализуйте следующие функции преобразования матриц:
   1. void transposeSquareMatrix(Matrix m) - транспонирует квадратную матрицу m.

void transposeSquareMatrix(Matrix m) {  
 **assert**(isSquareMatrix(m));  
  
 for (int i = 0; i < m.nRows; i++)  
 for (int j = i + 1; j < m.nCols; j++) {  
 int t = m.values[i][j];  
 m.values[i][j] = m.values[j][i];  
 m.values[j][i] = t;  
 }  
}

void testTransposeSquareMatrix() {  
 Matrix m = createMatrixFromArray((int[]) {1, 2, 3,  
 4, 5, 6,  
 7, 8, 9}, 3, 3);  
  
 transposeSquareMatrix(m);  
  
 Matrix expected = createMatrixFromArray((int[]) {1, 4, 7,  
 2, 5, 8,  
 3, 6, 9}, 3, 3);  
  
 **assert**(areTwoMatricesEqual(m, expected));   
  
 freeMemMatrix(m);  
 freeMemMatrix(expected);  
}

* 1. 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;  
}

void testTransposeMatrix() {  
 Matrix m = createMatrixFromArray((int[]) {1, 2, 3,  
 4, 5, 6}, 2, 3);  
  
 transposeMatrix(&m);  
  
 Matrix expected = createMatrixFromArray((int[]) {1, 4,  
 2, 5,  
 3, 6}, 3, 2);  
  
 **assert**(areTwoMatricesEqual(m, expected));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expected);  
}

1. В библиотеке matrix реализуйте функции для поиска минимального и макcимального элемента матрицы:
   1. Position getMinValuePos(Matrix m) - возвращает позицию минимального элемента матрицы m.

Position getMinValuePos(Matrix m) {  
 **assert**(m.nRows >= 1 && m.nCols >= 1);  
 Position minValuePosition = {0, 0};  
  
 for (int i = 0; i < m.nRows; i++)  
 for (int j = 0; j < m.nCols; j++)  
 if (m.values[i][j] < m.values[minValuePosition.rowIndex][minValuePosition.colIndex])  
 minValuePosition = (Position) {i, j};  
  
 return minValuePosition;  
}

void testGetMinValuePos() {  
 Matrix m = createMatrixFromArray((int[]) {19, 2, 3,  
 4, 1, 6}, 2, 3);  
  
 Position resultMinPos = getMinValuePos(m);  
  
 **assert**(resultMinPos.rowIndex == 1 && resultMinPos.colIndex == 1);  
  
 freeMemMatrix(m);  
}

* 1. Position getMaxValuePos(Matrix m) - возвращает позицию максимального элемента матрицы m.

Position getMaxValuePos(Matrix m) {  
 **assert**(m.nRows >= 1 && m.nCols >= 1);  
 Position maxValuePosition = {0, 0};  
  
 for (int i = 0; i < m.nRows; i++)  
 for (int j = 0; j < m.nCols; j++)  
 if (m.values[i][j] > m.values[maxValuePosition.rowIndex][maxValuePosition.colIndex])  
 maxValuePosition = (Position) {i, j};  
  
 return maxValuePosition;  
}

void testGetMaxValuePos() {  
 Matrix m = createMatrixFromArray((int[]) {3, 2, 19,  
 4, 1, 6}, 2, 3);  
  
 Position resultMinPos = getMaxValuePos(m);  
  
 **assert**(resultMinPos.rowIndex == 0 && resultMinPos.colIndex == 2);  
  
 freeMemMatrix(m);  
}

1. Дополните библиотеку функциями для тестирования:
   1. Matrix createMatrixFromArray(const int \*a, int nRows, int nCols) - возвращает матрицу размера nRows на nCols, построенную из элементов массива a:

Matrix createMatrixFromArray(const int \*a, int nRows,  
 int nCols) {  
 Matrix newMatrix = getMemMatrix(nRows, nCols);  
  
 int k = 0;  
 for (int i = 0; i < nRows; i++)  
 for (int j = 0; j < nCols; j++)  
 newMatrix.values[i][j] = a[k++];  
  
 return newMatrix;  
}

void testCreateMatrixFromArray() {  
 Matrix m = createMatrixFromArray(  
 (int[]) {  
 1, 1, 0,  
 0, 0, 0,  
 0, 0, 1,  
 0, 0, 0,  
 0, 1, 1,  
 }, 5, 3);  
  
 **assert**(m.values[0][0] && m.values[0][1] && !m.values[0][2] &&  
 !m.values[1][0] && !m.values[1][1] && !m.values[1][2] &&  
 !m.values[2][0] && !m.values[2][1] && m.values[2][2] &&  
 !m.values[3][0] && !m.values[3][1] && !m.values[3][2] &&  
 !m.values[4][0] && m.values[4][1] && m.values[4][2]);  
  
 freeMemMatrix(m);  
}

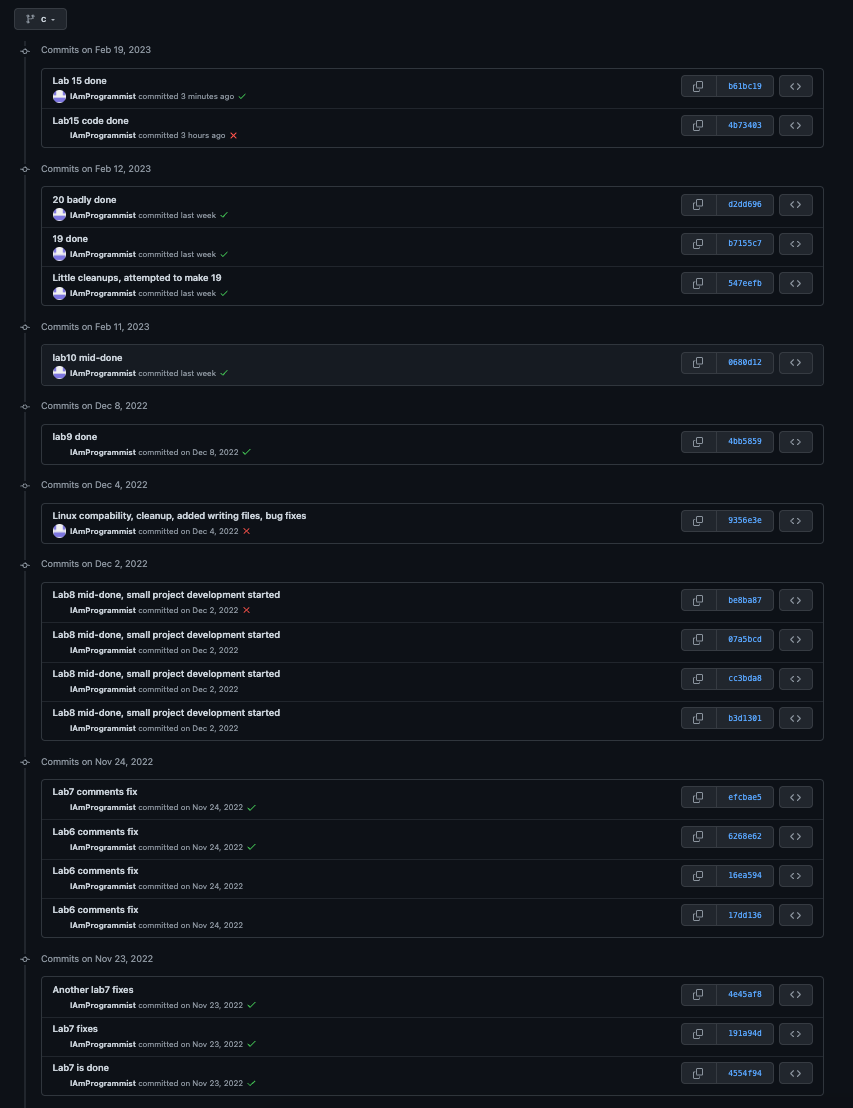
* 1. Matrix \*createArrayOfMatrixFromArray(const int \*values, size\_t nMatrices, int nRows, int nCols) - возвращает указатель на нулевую матрицу массива из nMatrices матриц, размещенных в динамической памяти, построенных из элементов массива a.

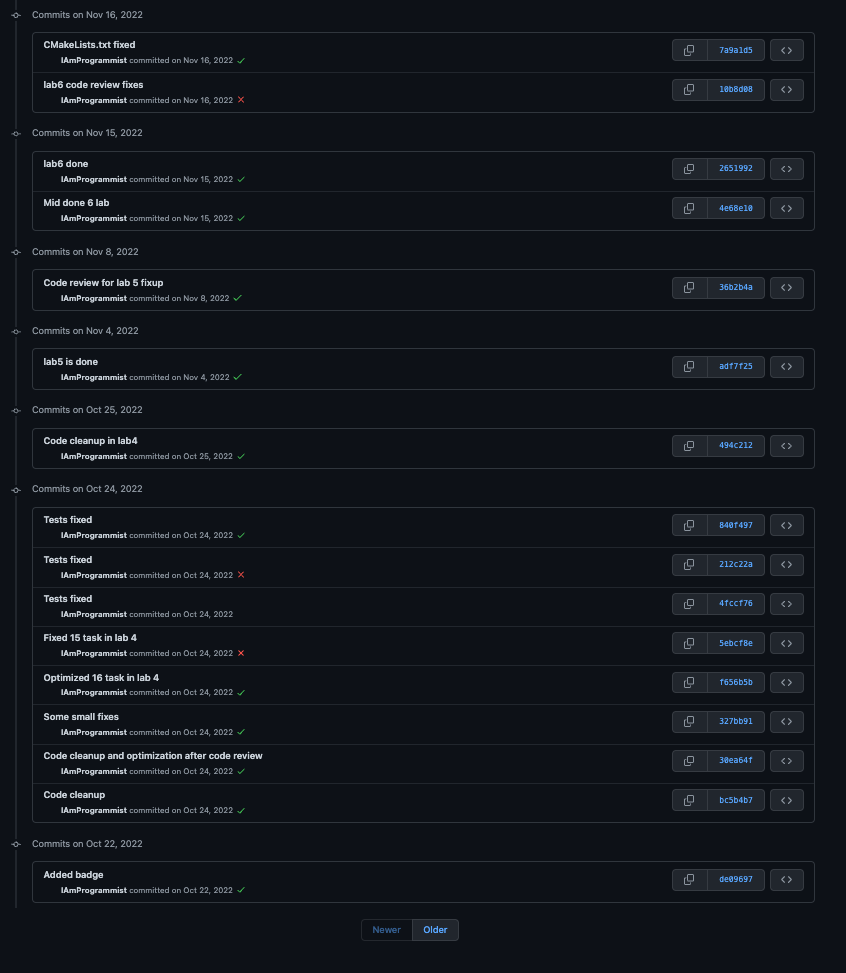
Matrix \*createArrayOfMatrixFromArray(const int \*values,  
 size\_t nMatrices, int nRows, int nCols) {  
 Matrix \*matrixArray = (Matrix \*) malloc(sizeof(Matrix) \* nMatrices);  
 int k = 0;  
 for (int i = 0; i < nMatrices; i++)  
 matrixArray[i] = createMatrixFromArray(values + (k++) \* nRows \* nCols, nRows, nCols);  
 return matrixArray;  
}

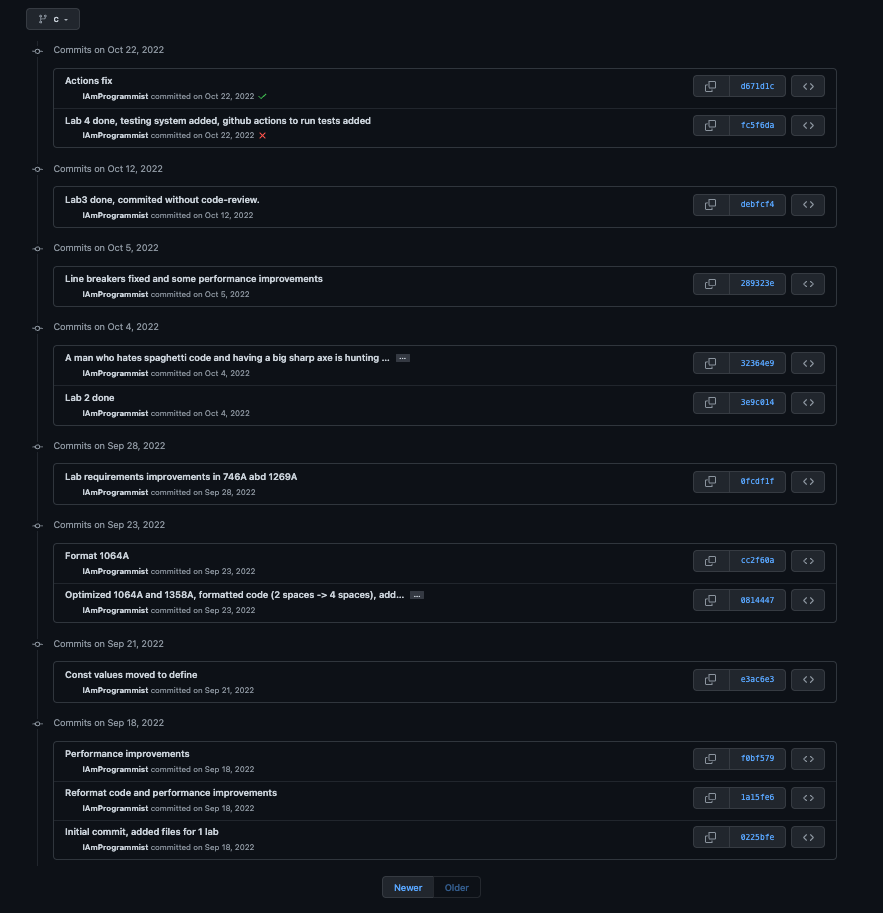
void testCreateArrayOfMatrixFromArray() {  
 Matrix \*mArray = createArrayOfMatrixFromArray(  
 (int[]) {  
 1, 1, 0,  
 0, 0, 0,  
 0, 0, 1,  
 0, 0, 0,  
 0, 1, 1,  
 1, 0, 1  
 }, 3, 2, 3);  
  
 **assert**(mArray[0].values[0][0] && mArray[0].values[0][1] && !mArray[0].values[0][2] &&  
 !mArray[0].values[1][0] && !mArray[0].values[1][1] && !mArray[0].values[1][2]);  
  
 **assert**(!mArray[1].values[0][0] && !mArray[1].values[0][1] && mArray[1].values[0][2] &&  
 !mArray[1].values[1][0] && !mArray[1].values[1][1] && !mArray[1].values[1][2]);  
  
 **assert**(!mArray[2].values[0][0] && mArray[2].values[0][1] && mArray[2].values[0][2] &&  
 mArray[2].values[1][0] && !mArray[2].values[1][1] && mArray[2].values[1][2]);  
  
 freeMemMatrices(mArray, 3);  
}

Ссылка на репозиторий: <https://github.com/IAmProgrammist/programming-and-algorithmization-basics>

Скриншоты с коммитами (репозиторий вёлся с сентября, в него выкладывалось решение каждой из лаб):







**Вывод:** в ходе выполнения лабораторной работы закреплены навыки создания библиотек, структур; получены навыки работы с многомерными массивами