Object Oriented Programming

Fall 2019

LAB-1

(issue date: September 21, 2020)

The objective of this lab is to:

1. Practice 2-D arrays.

Instructions!

1. This is an individual lab, you are strictly NOT allowed to discuss your solution with your fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can ONLY discuss with your TAs or with me.
2. Comment each line of your code.
3. Your TAs will be available online from 5:30pm to 6:30pm for help. Alternatively, you can send them your queries via email.
4. Submit your solutions on google classroom before mid-night. Late submissions will result in penalty.

TASK:

We will be creating a set of functions to facilitate us with matrix operations such as the allocation and de-allocation of matrices. When we are making an application or doing image processing, we do not want to worry about allocation of memory and memory leaks and want accessible functions.

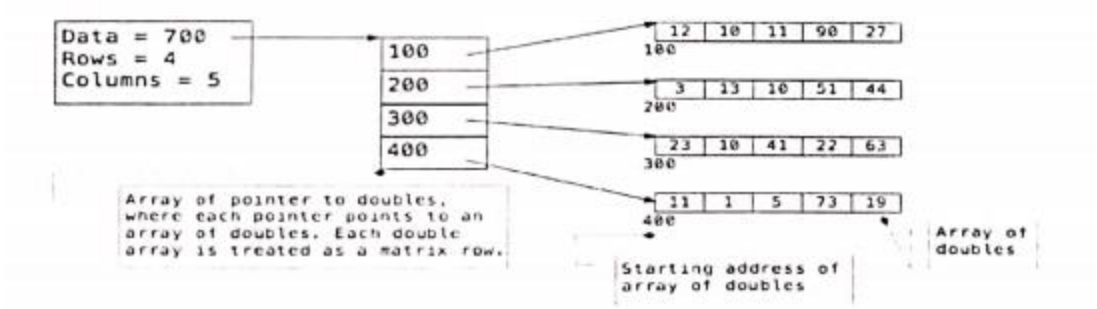
A library that supports all of the operations is more desirable.

Implement the following functions:

`void createMatrix (int*** m, int& rows, int& cols)`

dynamically allocate a 2d array to pointer m according to the number of rows and columns provided. In case of an error, don't allocate any memory, place null in *m and set the size to 0, 0.

If 4, 5 is passed as rows and cols, the memory layout will look like this



bool isIdentity (int p, int rows, int cols)**

Check if given matrix is an identity matrix

void display (int p, int rows, int cols)**

Display the matrix on console in a neat way

bool isLowerTriangular (int p, int rows, int cols)**

Check if given matrix is a lower triangular matrix

bool isUpperTriangular (int p, int rows, int cols)**

Check if given matrix is a upper triangular matrix

Bool isTriangular(int p, int rows, int cols)**

Check if given matrix is a triangular matrix

int getMatrixCopy (int** p, int row, int col)**

Return a copy of the matrix by re-allocation

bool isEqual(int a, int row1, int col1, int** b, int row2, int col2)**

Check if the two matrices are equal. Matrices are equal if their number of rows and columns are equal and all entries of the matrices are equal

int transpose (int** p, int row, int col)**

Transpose the matrix, a transpose is done by switching the rows with columns and columns with rows

void reSize (int* p, int& row, int& col, int newRows, int newCols)**

Resize the matrix in pointer p with the newRows and newCols if possible. Copy the elements from old matrix to new and update the variables row and col to reflect new size. If it is not possible, do nothing.

int multiply(int** mat1, int rows1, int cols1, int** mat2, int rows2, int cols2, int& resultRows, int& resultCols)**

multiply the two matrices and return the third matrix

Note: if multiplication is not possible then simply return null.

bool isSymmetric(int mat1, int rows, int cols)**

if A transpose = A return true, otherwise return false.

bool isSkewSymmetric(int mat1, int rows, int cols)**

if A transpose = -A return true, otherwise return false.

`void freeMatrix (int*** p, int row, int col)`

Deallocate the memory and set the pointer p to null

Sample Run:

```
int rows = 3;
int cols = 3;
int** mat;
createMatrix(&mat, rows, cols);
mat[0][0] = 0; mat[0][1] = 26; mat[0][2] = 3;
mat[1][0] = 2; mat[1][1] = 5;  mat[1][2] = 7;
mat[2][0] = 3; mat[2][1] = 7;  mat[2][2] = 0;
cout << "Identity: " << isIdentity(mat, rows, cols);
cout << "\nLowerTiang: " << isLowerTriangular(mat, rows,
cols);
cout << "\nUpperTriang: " << isUpperTriangular(mat, rows,
cols);
cout << "\nTriangular: " << isTriangular(mat, rows, cols);
int** cpy = getMatrixCopy(mat, rows, cols);
int cpyrows = rows, cpycols = cols;
cout << "\nEqual: " << isEqual(mat, rows, cols, cpy,
cpyrows, cpycols) << "\n";
display(cpy, rows, cols); cout << "\n";
```

```
Identity: 0
LowerTiang: 0
UpperTriang: 0
Triangular: 0
Equal: 1
0       26      3
2       5       7
3       7       0

0       26      3       0
2       5       7       0
3       7       0       0
7       8       9       0
```

```
reSize (&cpy, cpyrows, cpycols, 4, 4);
cpy[3][0] = 7; cpy[3][1] = 8; cpy[3][2] = 9;
display(cpy, rows, cols); cout << "\n";
freeMatrix (&cpy, rows, cols);
```