**Lab 2: Matrix Operations**

**Objectives**

In this lab, you will, will have an opportunity to work with two-dimensional arrays by defining some common matrix operations.

The matrix operations that we will write are on fixed, 3x3 square matrices, which should help eliminate some of the edge cases and complexity that arise in needing to deal with non-square matrices (3x4, 2x1, etc).

As always, if you are hazy on the mathematical notion of a matrix, go ahead and read up on them [here](https://github.com/Welchd1/cpsc210-labs/blob/master/labs/lab7/matrices.pdf) **Note:** Please read through all instructions carefully. You are responsible for handing in all material on time and correctly as well as following grading criteria below.

**Preparing for today's lab:**

Create a lab2 directory and *cd* to this directory. Use the following command to copy the files for this lab:

lab1071copy 2

List your files (*ls*). Your directory should now contain the files:

*makefile*, *matrix.c*, *matrix.h*, *matrixtest.c*, and *output.txt*.

**Task: implement 3x3 matrix operations**

We will represent our matrices with 3x3, 2D arrays of type double. Here are signatures for some of the operations you are expected to implement; the rest are defined in matrix.h.

/\*

\* Given matrices m1 and m2, compute m1 - m2 and put the resulting

\* matrix into 'result'.

\*/

void matrix\_difference(double m1[3][3], double m2[3][3], double result[3][3]);

/\*

\* Given matrices m1 and m2, compute m1m2 and put the resulting matrix

\* into result.

\*

\* Note that this is \*not\* simply multiplying each element in m1 by each

\* in m2!

\* It's more like taking the dot product between each row of m1 and

\* column of m2. Read up on it [matrices.pdf](https://github.com/Welchd1/cpsc210-labs/blob/master/labs/lab7/matrices.pdf)

\* Note that this operation \*replaces\* the incoming m.

\*/

void matrix\_transpose(double m[3][3]);

/\*

\* Returns zero if matrix m is symmetric; any other number otherwise.

\* Note that this operation should restore m, meaning by the time the

\* function ends, m should be the same as its incoming value.

\*/

int matrix\_is\_symmetric(double m[3][3]);

**2D Array Notation**

In this lab, we will use 2D array notation. We have seen 1D array notation before.

// 1D array example double a[5] = {0, 1, 2, 3, 4};

In order to represent a matrix more naturally, we will construct 2D arrays. To construct a 2D array we add another bracket to denote the additional dimension.

// 2D array example

double m[3][3] = {

{ 0, 1, 2 },

{ 4, 5, 6 },

{ 7, 8, 9 }

};

Take note of the above creation of a 2D double array. We can imagine the 2D array as an array of 1D arrays. To access a particular element from *m* after we have created it, we must use valid indices for accessing *m*. We denote the particular row and column that we wish to receive by writing *m[i][j]*, where *i* denotes the row and *j* denotes the column starting at index 0. So, *m[0][0]* accesses the top left element and *m[2][2]* accesses the bottom right element. We can also set an element in *m* by using the assignment operator = like this *m[1][1] = 0*. This sets the middle element in the matrix to 0. In order to access or set each element in a matrix we will use doubly and triply nested for-loops as discussed in the next section. For specific matrix operation help, please use [A Brief Introduction to Matrix Operations.](https://github.com/Welchd1/cpsc210-labs/blob/master/labs/lab7/matrices.pdf)

**Using a doubly or triply nested for-loop**

So far in our labs, we have dealt with a single nested for-loop. This is ideal for situations where we are dealing with 1D arrays. If we need to perform operations on 2D arrays, like matrices, we can nest for-loops to access every element in the array.

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

// result[i][j] = ...;

}

}

In the code above, we create a doubly nested for-loop. When accessing a 2D array using this method, we use the outer loop to denote the row we want to access. The inner loop we use to access the column of interest. Thus, we use the notation result[i][j] to set the ith row and jth column of a matrix result.

**Note:** when dealing with matrices, we read them from left to right, and top to bottom starting at index 0. The top-left is row 0, column 0, and bottom right is row 2, column 2.

We may need to go a step further and use a triply nested for-loop like in matrix\_multiply() to access a row from m1 and a column from m2. To do this we write the loop as follows:

double sum = 0.0;

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

sum = 0.0;

for(int k = 0; k < 3; k++) {

// sum += ...;

}

}

}

Remember that when going through the for-loop, the innermost loop will complete first (k loop), then the next loop (j loop), and finally the outermost loop. We continue going through the loop until the outermost loop runs 3 times.

**Note:** use the fact that we want to use the corresponding row of *m1* and column of *m2* as a hint to construct your assignment.

**Testing**

To compile your code, type make. This will build the code and generate an executable for it automatically. Then to run, all you need to do now is type ./lab2. Also, remember that you can freshen your working directory -- deleting any lingering \*.o files -- by typing make clean.

The expected output can be viewed in [output.txt](https://github.com/dtwelch/cpsc210-labs/blob/master/labs/lab7/src/output.txt).

**Handin**

When you're finished, and you are confident your work is adequately commented and correct, do the following:

1. use *make clean* to remove all object (.o) files and lab2
2. 'tar' the files using the with the following command:

tar -cvf lab2\_handin.tar \*

1. submit the resulting lab2\_handin.tar to the appropriate bucket on [handin](https://handin.cs.clemson.edu/)

**Due Date**

As always labs are due the day after your assigned lab section by 11:59:59pm.

**Grading**

You will be graded using the following criteria:

* Correctness of output for each function.
* Use of double/triple nested for-loops for solutions. (Note: only matrix\_multiply uses a triple for-loop)
* Proper use of comments. (Note: you do not need to over comment; we are just looking for a brief explanation of your code)
  + Comments are also very helpful if your function does not work because we offer partial credit if your logic is correct.
* Proper use of spaces/indentation. (Note: look at the indentation noted below; do not left-align all of you code)
  + You do not need to follow this exact level of indentation, this is just a template.
  + Just make sure that your indentation is consistent.
* Logical use of variable names
  + ex: don't name variables x, something, not\_sure, etc.
  + It's ok to use i, j, and k as variables in for loops.
  + Use some naming scheme that provides context to your variable
* Handing in correct materials
  + matrix.c, matrix.h, matrixtest.c, makefile, and output.txt inside lab7\_handin.tar (Note: use command from above)
* On-time submission
* Lab attendance

**Indentation Example**

#include

int main()

{

double m[3][3] = {

{0, 0, 0},

{0, 0, 0},

{0, 0, 0}

};

for(int i = 0; i < 3; i++)

{

for(int j = 0; j < 3; j++)

{

m[i][j] = 1;

}

}

return 0;

}

**Final Note**

At any point during or after the lab, do not hesitate to ask for help or send an email to your lab instructor.