ETEC2110	Lab8: **Structs and multi-dimensional arrays**	Points: **52**

1. (**10 points**) Get a working copy of Lab7. Convert our "parallel" arrays (id's, float's, and string's) into an array of structures, but otherwise keep the same general layout as we had in Lab7. You should only have one array of structures that is created in main. Ask if you want some early feedback on whether you're converting things the way I'm looking for. Note: You will have to make a temporary copy of your float salary values to compute stats.
2. Matrix math
    a. *Note: if you've not seen matrices before, I've put together a very brief primer on the second page (and as always, ask if you have other questions)*
    b. (**5 points**) Do the project re-organization we discussed in class. (**+6 points** if you support all 3: code::blocks, command-line makefile, and Visual Studio)
    c. Make a matrix_math.h
        i. Typedef and declare a **Matrix** struct that has two unsigned integers (number of rows and columns) and a float data which will eventually be a dynamically allocated 2D array of floats.
        ii. Declare all the functions from matrix_math.c
    d. Make a matrix_math.c file (no global variables!)
        i. (**5 points**) Matrix matrix_create(unsigned int rows, unsigned int columns): Makes a matrix structure and dynamically allocates the internal 2D array. Note that returning this way is valid, but now the caller is given the responsibility of ensuring that matrix_destroy is called to free up the memory. Initialize all the elements to 0.0f after creating.
        ii. (**3 points**) void matrix_destroy(Matrix* m): frees up the memory of the passed matrix – set the number of rows and columns to 0 (and set the pointer to null) to indicate this is no longer a valid matrix.
        iii. (**3 points**) void matrix_set_item(Matrix* m, unsigned int row, unsigned int column, float value): Sets an element of the internal 2D array (if it's invalid, do nothing)
        iv. (**3 points**) float matrix_get_value(Matrix* m, unsigned int row, unsigned int column): return the value at the spot (or NAN if not)
        v. (**3 points**) void matrix_set_values(Matrix* m, float* all_values): Takes the values in a passed flat array of values and use it to set all values in the array. It is the caller's responsibility to ensure this array is of the proper size. This function should call matrix_set_item to actually change the matrix.
        vi. (**7 points**) void matrix_print(Matrix* m, FILE* outfile): Write a matrix in this format to the given file (remember that the user can pass stdout if they wish to output to the console):

```
/a      b      c\
|d      e      f|
\g      h      i/
```

If there is only one row, do this type of output instead

```
[a      b      c]
```

If there are only two rows, ignore the | in the first sample. This function is to use matrix_get_value to get the matrix values.

(**7 points**) Matrix matrix_multiply(Matrix* a, Matrix* b): If the matrices are of incorrect shape to be multiplied, return a "invalid" matrix (0 number of rows and columns, null data pointer). Otherwise, create and return the result of multiplying the matrices.

    e. (**6 points**) Make a main program that creates a few matrices and outputs the result. I used C = A * B and D = B * A (matrix A and B are defined on the next page) and got this:

```
A =
/5.000000       -3.000000       8.000000        1.000000\
|2.000000       -1.000000       5.000000        4.000000|
\0.000000       9.000000        3.000000        -2.000000/
B =
/8.000000       7.000000\
|-3.000000      2.000000|
|6.000000       3.000000|
\0.000000       -5.000000/
C =
/97.000000      48.000000\
|49.000000      7.000000|
\-9.000000      37.000000/
D is a null-matrix (as expected)
```

Matrices are a very powerful tool used in Linear Algebra and other domains. Basically, a matrix is a collection of values. It is not important for purposes of this lab, but those values usually represent coefficients of a set of linear equations. For example, if we have these equations:

5x – 3y + 8z = 24
9y – z = 14
-2x + 5y + 17z = 32

We could represent it in matrix form as this (technically this has three matrices, but the 3x3 one is the one interest):

$$\begin{bmatrix} 5 & -3 & 8 \\ 0 & 9 & -1 \\ -2 & 5 & 17 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 24 \\ 14 \\ 32 \end{bmatrix}$$

Again, not important for this lab, but there are many operations we can perform on matrices: inverses, eigen-decomposition, etc. All we are going to focus on in this lab is constructing a matrix and multiplying it by another.

First, some terminology: one can describe a matrix's shape by defining the number of rows and columns. For example, the following matrix is a 3x4 (3 rows, 4 columns):

$$A = \begin{bmatrix} 5 & -3 & 8 & 1 \\ 2 & -1 & 5 & 4 \\ 0 & 9 & 3 & -2 \end{bmatrix}$$

An individual element of a matrix can be described with this notation (note: I'm using 0-based "indexing" – many mathematicians use 1-based "indexing"):

$A_{0,0}$ = 5
$A_{1,0}$ = 2
$A_{2,3}$ = -2

Two matrices, A and B, can be multiplied if the number of columns of A matches the number of rows of B (note that matrix multiplication is not commutative). The resulting matrix C will have number of rows equal to A's number of rows and columns equal to B's number of columns.

If the above matrix is called A, we know that B (if to be multiplied by A) needs to have 4 rows). One such matrix might be:

$$B = \begin{bmatrix} 8 & 7 \\ -3 & 2 \\ 6 & 3 \\ 0 & -5 \end{bmatrix}$$

Since we have this situation:

$A_{3x4} * B_{4x2}$

We will obtain a new matrix C, which is 3x2. Each element of C can be obtained this way:

$$C_{i,j} = \sum_{k=0}^{p} A_{i,k} * B_{k,j}$$

Where p = the number of columns in A (and also the number of rows in B) and i = a row in C, and j = a column in C.