ETEC2110	Lab7: **Dynamic memory allocation + basic File I/O**	Points: **40**

1. On this lab, you can choose whether to put the functions in a separate "module" (.h/.c file) or directly in the main program [they're not super general-purpose; when a function is, I personally prefer to break it out into its own module]
2. Do not use any global variables in this program. Some of you have been sneaking these in to "cheat" on return values. There is a place for globals, but we haven't seen it yet.
3. Write (at least) these 3 functions to aid in the parsing of a text file like that on blackboard. But keep in mind, that file is just a sample – make no assumption about the number of records (you *can* assume the general structure of individual lines will be the same)
    a. (**7 points**) **get_datafile_size**. Should take a string as an argument. The function should open the file and count the number of data-containing lines and return it. Data-containing is defined as a non-empty line that doesn't start with a '#' and that contains exactly two colons.
        i. You can assume all names in the file will be less than 30 characters – encode this assumption in a macro value
    b. (**10 points**) **read_datafile**. Should take a string file-name as in get_datafile_size. Should also take a pointer to three arrays of the size appropriate to hold all data in the file. The first array should be a flat (no multi-dimensional arrays yet) array of strings, the second an array of integers (to hold the ids) and the last an array of floats to hold the salaries. Note: the array for the names is technically [macro value from 3a] * number_of_valid_lines. I want the first string to start at character 0. The second string should start at character [macro value]. The third should start at character [macro value * 2], and so on.
        i. I want this function to read the file line-by-line (using fgets) and pass control to the 3rd function to actually parse the line.
    c. (**10 points**) **extract_employee_data**. This function should take a line (from the text file in this case) and break it into 3 parts, assuming we have a valid line (if not, return 0). I want you to pass 3 by-pointer parameters from elsewhere (read_datafile in this case) and this functions fills those in. Return 1 if we successfully parse the line. As an exercise, I'd like you to use strtok to break the line into parts. Return a 1 if we successfully parsed a line, 0 if not.
4. Incorporate your lab6_util module's **get_stats** function (or use mine from the solution)
5. (**13 points**) The main program should call get_datafile_size, allocate the 3 arrays of sufficient size, then call read_datafile. Finally, print out the contents of the arrays (I want this to happen in main or another function, not when you read it from the file). Print out all the values, but display a "<" before the name if this person makes below the average salary, ">" if they make above and "<<" or ">>" if they make either the min or max. Use your lab6_util function to get the min, max, and average. Finally, free up the arrays.
    a. Note: if you do "%-20s" and pass a string, the – tells C to right-justify – do that for name
6. (**+8 points**) Make this an interactive program like we did in class (show a menu and let the user delete, add, modify and list the values)
7. (**+8 points**) Make an implementation of bubble sort (https://en.wikipedia.org/wiki/Bubble_sort) that sorts all records by last name, but otherwise produces the same results.