COP 3502 Suggested Practice Programs: Dynamic Memory Allocation

1) Write a program that has a few functions that perform the following tasks:

a) A function that takes in an integer n, representing the number of items, and an integer, max, representing the maximum possible item, dynamically allocates an integer array of size n where each number is randomly selected in between 1 and max, inclusive, and returns a pointer to this newly created array.

b) A function that takes in an array and its size, and prints out the contents of the array.

c) A function that takes in an array and its size, and returns the maximum value in the array.

d) A function that takes in an array and its size, and returns the average of the values in the array.

Have your program call the function to generate the array, and then, with this array, print it, its maximum value and its average. Feel free to add other features to the program. At the end of the program, free the memory for the array.

2) Design your own struct and dynamically allocate an array of that struct and populate the array with values. Write functions that take in the array and its length, and return some information about the array. (For example, if your structs were cars, you could have a function that returns the speed of the fastest car.) At the end, free the array.

3) Redo #2 but with an array of pointers to structs, where each struct is individually allocated. In this version, write a function that returns a pointer to the struct, where the job of the function is to ask the user some questions and return a pointer to the struct they created.

4) Ask the user to enter a list of strings and dynamically allocate the space for the array. After the user enters the strings, sort them using the appropriate string functions and any sorting algorithm or implementation you can devise.

5) Write code to read in two big integers and store each in a dynamically allocated integer array, where each element stores a digit. It makes sense to store the last digit (units digit) in index 0, the next to last digit (tens digit) in index 1, and so forth. Write a function that takes in pointers to two arrays storing integers in this fashion and returns a pointer to a newly allocated array that stores a big integer that represents the sum of the two values.

6) Write a function that takes in one big integer as described in problem #5 and a single digit and then multiplies the single digit by the big integer, allocates memory for a new integer array and stores the product of the two values in this new array and returns a pointer to it.

7) Write another function that takes in a big integer as previously described in problem #5, and a positive integer, shift, and returns a new big integer that is the input big integer multiplied by 10^{shift}. In essence, this function should add shift number of 0s to the end of the big integer.

8) Uses programs #5, #6 and #7 to build a function that multiplies two big integers as stored in problem #5, and create a program that reads in large integers to multiply, multiplies them, displaces the result and frees all of the associated memory.

9) Write a program that reads in two matrices (2D array) of integers from a file and dynamically allocates memory to store both matrices. Write a function that takes in pointers to both matrices and allocates memory to store the matrix product, calculates this and returns a pointer to the new 2D array. Free all the associated memory after displaying the result for the user.

10) Write a program that reads in a dictionary of words. Find a file where the first line stores the number of words and the following lines have one word each and use this for testing. Your program should dynamically allocate an array of strings to store all the words. Then, go through the whole array and count how many words start with each letter. Store these values in an integer array of size 26. Finally, create a three dimensional array dynamically, where the first dimension has size 26 (where the index represents the starting letter of words) and the second dimension represents the number of words that start with the corresponding letter. The third dimension should be variable based on the length of the individual word being stored. Copy the contents of the original file into this 2D array of strings so that index 0 stores all words that start with 'a', index 1 stores all words that start with 'b', and so forth. Free the original array storage. Then, allow the user to enter in a starting letter and list all of the words in the dictionary that start with that letter. At the end, free the whole 3D array structure. Feel free to add any features that seem useful.