John Gill

Dr. Tala Talaei Khoei

CS 5008

2 June 2024

Lab Activity 2

I found this lab activity’s problems to be both interesting and challenging in a satisfying way. The book library initializes 4 void functions: addBook, displayBooks, totalBooks, and findBook. “AddBook” contains a book library structure. It asks the user for a title, an author, and a printing year, then saves all those values to a newBook variable, and adds it to the growing book library structure, then increases numBooks by 1. “DisplayBooks” iterates over the current bookLibrary and prints each value. “TotalBooks” prints the current value of numBooks, and findBook asks the user for a book title and checks if it’s stored in the Book library structure. Then, the main function initializes a Book library with a max value of 100, a numBooks integer set to 0, and a choice integer with a five-case switch to call on any of the 4 void functions, or exit.

The next two problems seem similar at first, but I came to discover their only similarity was in the fact that they use matrices. The first of the next two has us rotate a 3x4 matrix 90 degrees clockwise. That was done by flipping the value locations of the matrix across a diagonal axis spanning from the top-left corner to the bottom-right, then flipping that resulting matrix across the y-axis to print in that order. NOTICE in the screenshot I post for that question, it says “2” where it should say “12.” I think that is just a bug with my PowerShell, because everything about how the code works should correctly print the entire array. The second of the next two required a 2D matrix to be printed in diagonal order, with each diagonal on their own line, from bottom-left to top-right. I achieved this code by making a for loop based on the sum of rows and columns, called “diagonalSum”, and printing each value while iterating through it by lowering the start row and raising the start column of each coordinate.

The third problem was fairly straightforward. It initializes a character pointer that points to an alphabet character array, then creates a while loop that prints out each character while iteratively increasing the array coordinate until it returns NULL.

Code 1:

A computer screen shot of text

Description automatically generated

This code starts by initializing a void function that takes as an argument a pointer to a pointer to a string variable. The main function initializes an array of character pointers, then runs it through the void function located at the bottom of the code. The void function at the bottom takes “(char \*\*ptr)” as a single parameter, which is a pointer to a point to an integer variable, named “ptr1.” The ptr1 variable equals the size of (int) minus 2. The size of int in C refers to the value of size of an integer: 4. Thus, ptr1 equals 4 minus 2, to 2. Thus, in the main function, the array of character strings points to spot 2, the third value, which is “cat”, and that’s what gets printed.

Code 2:

A computer screen shot of a code

Description automatically generated

In the main function, a structure named “node” is defined with three integers: a, b, and c. Then, an instance of that node, named “num”, is initialized with values 3, 5, and 6. Then, a pointer to the node is declared and addressed “num.” Ptr itself points to the start address of num, which is the first spot, meaning it takes a value of 0. With ptr being 0, the “\*((int\*)” portion of the print statement points to a spot in num equal to ptr + 1 + (3-2), which is 0 + 1 + 1, equaling 2. The spot in num equal to 2 is the third spot, containing value “6”, meaning 6 is printed.

Code 3:

A screen shot of a computer program

Description automatically generated

This code initially didn’t work for me because the main function needs to be type “int” instead of type “void” on my computer. That being said, once I changed it to type “int”, the code’s output was 19. The way it works is by first setting an f function with integer variable “x”, a pointer (P) to variable “y”, and a pointer-to-pointer (PTP) to variable “z.” Y and z need to be initialized within the function since their heading nomenclature specifies them as pointers. The value given to PTP z increases by 1, then int z becomes that value. The same happens with P y and int y, respectively, except they increase by 2, with int x following that trend as well for an increase of 3.

Once in the main function, we initialize an int “c”, a P “b”, and a PTP “a.” A points to b as being equal to it, as does b to c. C is set to 4, and with b and c both being pointers to a set variable, they will each change together when that variable passes through the f function. C in the main function is x in the f function, with b being y, and a being z. Since the other two variables point to c in the main function, we start incrementing 4 by the value associated with the furthest pointer from it. That furthest pointer is a, associated with z in the f function, meaning 4 increases by 1 up to 5 for both the a and b variables. Now that b is 5, when it goes through the f function as y, it will increase from 5 by 2 to 7. Then c, still with its set value of 4, goes through the f function as x, and is increased by 3 to 7. The resulting variables, when added together, are 5 + 7 + 7, which equals 19.