CSE31: Lab #3 – Memory in C

Overview

In this lab, we will investigate some interesting facts about how C interacts with memory. You will continue to practice the use of pointers to access memory.

You can refer to chapter 5 and 6 of K&R for references on pointers.

Getting started

Before we begin any activities, create a directory (Lab_3) inside the CSE31 directory we created during Lab #1. You will save all your works from this lab here.

(Exercise) How memory is used

From lecture, we've learned that there are 3 pools of memory for storing variables, based on the nature of their usage.

TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and do the following tasks (20 minutes):

1. Name the 3 pools for memory and what kind of variables will be stored in each pool.
2. Open ***mem.c*** with your favorite text editor and discuss the following questions with your partner:
    a. How many variables are declared?
    b. How many of them are pointers? What type of data does each pointer point to?
    c. Which pool of memory are these variables stored in?
    d. Which pool of memory will the pointer ***ptr*** point to in line 12?
3. Using a piece of paper (or a drawing app), draw the 3 pools of memory and indicate the locations (in which pool?) of the variables in ***mem.c*** using boxes (like what we did in lecture). Label the boxes with variable names, their content, and their addresses. You will need to insert extra code to obtain the addresses of these variables.
4. In the same drawing, use arrows to connect each pointer to its destination.
5. Show your drawing to you TA to verify if it is correct.

6. Your TA will “invite” one of you randomly to share what you have discussed.

(Exercise) Structures in C

In this exercise, we will explore how structures are stored in memory

TPS (Think-Pair-Share) activity 2 Paired with the same classmate and do the following tasks (20 minutes):

1. Open ***NodeStruct.c*** and discuss what this program does.
2. Insert extra code to print out <u>*addresses*</u> of ***head***, <u>*value*</u> of ***head***, <u>*addresses*</u> of ***iValue***, ***fValue***, and ***next*** pointed by ***head***.
3. Based on the addresses of the members of ***Node*** structure, what do you observe about how structures are stored in memory? What is the relationship between the pointer (***head***) and its destination (the ***Node structure***)?

(Assignment 1, individual) Arrays and pointers

As we have discussed in lecture, we can use array names as if they are pointers. Open ***array.c*** and complete the following tasks:

1. This program will store integers entered by a user into an array. It then calls ***bubbleSort*** to sort the array. Study the code in ***bubbleSort*** to refresh your memory on Bubble Sort algorithm and answer the following questions:
   a. Why do we need to pass the size of array to the function?
   b. Is the original array (the one being passed into the function) changed at the end of this function?
   c. Why do you think a new array (***s_array***) is needed to store the result of the sorted values (why not update the array as we sort)? Hint: look at what the ***main*** function does.
2. Once you remember how Bubble Sort works, ***re-write*** the code so that you are accessing the array’s content using **<u>pointer notations</u>** (****s_arr***). i.e. you cannot use ***s_arr[j]*** anymore. Comment out the original code so the algorithm won’t be run twice.
3. After the array is sorted, the program will ask user to enter a key to search for in the sorted array. It will then call ***bSearch*** to perform a Binary Search on the array. Complete the ***bSearch*** function so that it implements Binary Search ***recursively (no loop!)*** You must use **<u>pointer notations</u>** here as well. Pay attention to what is written in ***main*** so your ***bSearch*** will return an appropriate value.

(Assignment 2, individual) Cyclic Linked list

In ***cyclic_ll.c***, complete the function ***has_cycle()*** to implement the following algorithm for checking if a singly-linked list has a cycle.

Recall that if ***p*** is a pointer to a struct, ***p->member*** refers to a member variable in the struct, and is equivalent to ***(*p).member***.

1. Start with two pointers at the head of the list
2. On each iteration, increment the first pointer by one node and the second pointer by two nodes. If it is not possible to do one or both because of a null pointer, then we know there is an end to the list and there is therefore no cycle.
3. We know there is a cycle if
   a. The second pointer is the same as the first pointer
   b. The next node of the second pointer is pointed to by the first pointer

The reason we know there is a cycle with the two conditions in 3) is that second pointer has wrapped around to the first one in the circle and we have detected it. After you have correctly implemented ***has_cycle***, the program you get when you compile ***cyclic_ll.c*** will tell you that ***has_cycle*** agrees with what the program expected it to output.

What to submit

When you are done with this lab assignments, you are ready to submit your work. Make sure you have included the following ***<u>before</u>*** you press Submit:

- Your ***mem.c***, ***NodeStruct.c***, ***array.c***, ***cyclic_ll.c***, ***answers to the TPS activities/assignments*** in a text file, and a list of Collaborators.
- Your assignment is closed **7 days after this lab is posted** (at 11:59pm).
- You must demo your submission to your TA **within 14 days** (preferably during next lab so you will have a chance to make correction and re-submit/re-demo.)