Lab 03: Memory in C
CSE-031-01 **Points: 20**

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 last week. You will save all your work from this lab here. **Note that all the files shown in green below are the ones you will be submitting for this assignment.**

You must have a clear idea of how to answer the lab activities before leaving lab to receive participation score.

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: Discuss questions 1 – 4 (20 minutes) while paired with your classmates assigned by your TA (you will be assigned to groups of 3-4 students) and record your answers in a text file named `tpsAnswers.txt` under a section labelled "TPS 1" (***you will continue to use this file to record your answers to all the TPS questions that follow in the lab handout***):

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.

Your TA will "invite" one of you randomly after the activity to share what you have discussed.

Structures in C

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

TPS activity 2: Discuss questions 1 – 3 with your TPS partners in your assigned group (20 minutes) and record your answers in `tpsAnswers.txt` under a section labelled "TPS 2":

1. Open `NodeStruct.c` and discuss what this program does.
2. Insert extra code to print out the *value* of `head`, *addresses* of `head`, `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)?

Individual Assignment 1: 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 understand how Bubble Sort works, ***re-write*** the code so that you are accessing the array's content **using pointer notations** (`*s_arr`), i.e., you cannot use `s_arr[j]` anymore. Comment out the original code so the algorithm will not 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 **pointer notations** here as well. Pay attention to what is written in `main` so your `bSearch` will return an appropriate value.

Individual Assignment 2: 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.

Collaboration

You must credit anyone you worked with in any of the following three different ways:

1. Given help to
2. Gotten help from
3. Collaborated with and worked together.

What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Your code compiles and runs without the need for special libraries.
- Attached `mem.c`, `NodeStruct.c`, `array.c`, `cyclic_ll.c`, and `tpsAnswers.txt`.
- Filled in your collaborator's name (if any) in the "Comments…" textbox at the submission page.

Also, remember to demonstrate your code to the TA or instructor before the end of the grace period.