Andre Martin
Thursday 09L (4:30-7:20pm)
CSE 31

Lab 3

TPS (Think-Pair-Share) Activity 1:

1. Name the 3 pools for memory and what kind of variables will be stored in each pool.

- Static: local Variable
- Stack: dynamic memory
- Heap: global variable

2. a. How many variables are declared?

- In the program there are 3 variables declared.

b. How many of them are pointers? What type of data does each pointer point to?

- Num is an integer variable, ptr is a pointer variable that will hold the address of int and handle is a pointer variable to, which holds the address on pointer.

c. Which pool of memory are these variables stored in?

- num is stored in stack memory, ptr and handle in Heap Memory

d. Which pool of memory will the pointer *ptr* point to in line 12?

- It would point to heap memory

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.

TPS (Think-Pair-Share) activity 2

1. Open *NodeStruct.c* and discuss what this program does.

- This program makes a Node with a head pointer allocating memory inside the heap big enough for both the head and '*next' node and also setting the values inside the head's node to some values.

2. Insert extra code to print out *addresses* of *head*, *value* of *head*, *addresses* of *iValue*, *fValue*, and *next* pointed by *head*. Done

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*)?
- The program, a structure looks like it is stored as a memory block inside the memory and inside that memory block, there is enough space to include variables and values as the program does. The relationship between the head and the Node structure is that the head is representing such Node in this program with the head's pointer pointing at the structure.

Assignment 1

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?

- You need to pass the size of the array because without it you cannot travel up to the last element of array. If size is unknown one may go beyond the elements or may not travel till last elements. It avoids array out of bound exception.

b. Is the original array (the one being passed into the function) changed at the end of this function?

- Yes, original array will get altered. We now array name points to the address of first element of array. eg in a[50], a stores address of a[0]. So when array is passed to a function not values but address is passed. So, any changes gets reflected.

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.

- If a new array is not used the original array will be no more available after sorting due the fact above mentioned. A new array is needed when we store our sorted array so that the user can see the original array they've picked and compare it with the newly sorted array

2. Once you remember 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 won't be run twice. Done

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. Done

Assignment 2:

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

Collaborators: Omar Silva