TPS (Think-Pair-Share) activity 1: Discuss questions 1 – 4 (20 minutes) while paired with your classmates in a breakout room 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.
static: global variable storage, permanent for the entire run of the program.
stack: local variable storage (automatic, continuous memory).
heap: dynamic storage (large pool of memory, not allocated in contiguous order)

2. Open mem.c with your favorite text editor and discuss the following questions with your partner:

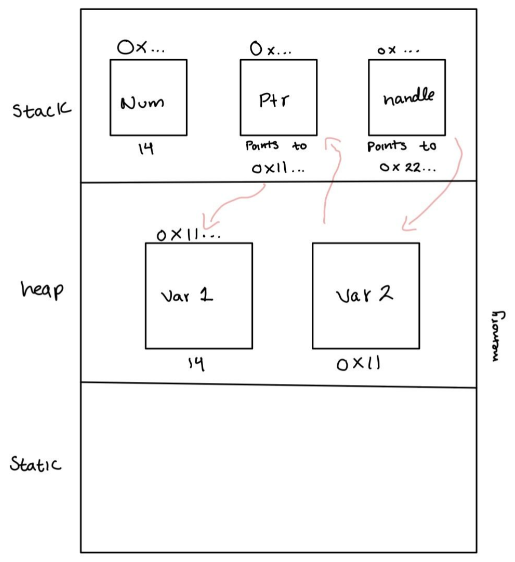
a. How many variables are declared?
There are 3 variables and 2 of them are pointers.

b. How many of them are pointers? What type of data does each pointer point to?
There are two of them and they both point of integers.
The handle is a pointer that is pointing to another pointer.

c. Which pool of memory are these variables stored in?
They are all stored in the stack since they are declared inside main().

d. Which pool of memory will the pointer ptr point to in line 12.
It will point to heap memory since its dynamic memory allocations.

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 activity 2: Discuss questions 1 – 3 with your TPS partners in your assigned breakout room (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.
Its a linked list.

2. Insert extra code to print out the value of head, addresses of head, iValue, fValue, and next pointed by the head.
The code is visible in the code file.

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)?

Structure in c allocates memory itself on basis of its data type, it calculates the sum of memory from its datatype memory requirement and it allocates memory which is the sum of memory of all data types. Head is a pointer to structure. In this problem, head is pointing to node structure; By using pointer structure we can allocate memory of structure dynamically.
Answers

Individual Assignment 1: Arrays and pointers As we have discussed in the 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?

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

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

- Yes, the original array will get altered. We know array name points to the address of the first element of array.
    - eg in a[50], a stores address of a[0].

- So when the array is passed to a function, not values but the address is passed. So any changes get 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 new array is not used the original array will be no more available after sorting due the fact above mentioned.

- if a points to adress of a[0] to get value stored in a[0] we can use the indirection operator *. Thus *a gives value stored inside a[0]. eg a[]={1,2,3,4,5}
- *a=>5 *a+1)=>2   *(a+2)=>3 ....etc

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 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.