Rutam Kathale October 12, 2023

CS-1203 Problem Set 1

Collaborators: *none*

Problem 2-1. Pointer Arithmetic

The idea of this problem is to get you familiar with pointers in C. Take a look at thefollowing code snippets, run them and tell us the output (in a verbatim block), and clearly justify the output. (a) On my machine, the following loop prints -1808407282 . Why does it stop?
Solution:

(b) What is the following code doing? What will be the output of this program if the user input abxyba?
Solution: The code essentially is printing all scanned characters together which aren't 'x' or 'y' basically stripping of x's and y's from the string. It achieves this by first scanning a string (less than or equal to 100 characters/bytes), and looping over the array and checking if the current character is x or y, if not it prints it so it will print abba.

(c) What will be the output of the following C program and why?
Solution: We essentially print the address of x, value of x, address of x in 3 separate lines. (While address can't be correctly represented using $\%d$, let's assume that it does just to focus on the main semantic idea).
We first declare an integer x and initialize it to 42. We then declare a pointer notblah and set it to point to x by giving it address of x using &.
So, we first print the address of x. Then we call the function doSomething and pass the pointer notblah.
We there change the value of x using $*$ which stands for 'value at' basically the value at whatever blah is storing which is address of x.
Then obviously the second line prints 420. Since we are calling by reference, the changes happen to original variables.
We print the same address in third line since the 'value at' x has changed but the 'address of' x remains the same.

(d) What will be the output of the following this program and why?
Solution: The code basically attempts to print the value of a variable wherein the value is manipulated by pointers. So, we first define 'a' and pointer 'ptr' and attempt to change the value of 'a' by passing the address of 'a' to the pointer and reassign 'a' using $*$. But since both 'a' and 'ptr' are constants, reassigning won't be possible and the compiler will throw an error.

(e) What will be the output of the following C program and why?
Solution: The following program attempts to reverse the given string of characters (25 characters or less) by basically swapping every ith element with every ith element (from the right) - 1st element from the left would be swapped with last element, 2nd element from left would be swapped with last second element, and so on. Technically speaking, the program won't run due to the wrong logic in indexing. That is, instead of putting last element in 0th index, it would attempt to put it in index of -1 which isn't possible. The correct way to index would be x[i] = s[length - i - 1] which would mimic the swapping mechanism except that we use a new array and put every element instead of swapping in-place.

(f) What will be the output of the following C program and why?
Solution: The below code basically prints the value of character 'a' (character-> integer) since we print the character as integer using %d. Then we pass a character 'A' in an int type variable and print it as an integer. Here, what gets printed is basically the ASCII value corresponding to 'A'.

(g) What will be the output of the following C program and why?
Solution: The following code demonstrates pointer arithmetic. We first initialize variable 'i' as 2984 and assign the pointer 'j' to the address of i. Then, we reassign i with i and the value at address of j which is, i itself. So, basically we end up doubling the value of i and then we print it. So, the output would be 5968.

(h) What will be the output of the following C program and why?
Solution: The code basically prints information about the sizes and addresses of data types. It initializes pointers to specific memory addresses (0x5000, 0x7000, 0x9000) and displays the size of each data type and the addresses of the initialized pointers and their respective addresses. It basically demonstrates the sizes and addresses of empty 'struct', 'char', 'int'.
So, we get outputs like:
int 0x01 0000000000005000 0000000000005001
int 0x04 0000000000007000 0000000000007004
struct 0x00 0000000000009000 0000000000009000

(i) What will be the output of the following C program and why?
Solution: In this code, we basically print a substring of "LATENCY". We achieve this by first get pointer 'p' to point to start of array. Then we use p[3] - p[1] ('E' - 'A') to get index 4 and we print array[4:] which results in 'NCY'.

(j) What will be the output of the following C program and why?
Solution: We basically initialize ‘p‘ as a pointer to the first element of ‘arr‘, and ‘pp‘ is a pointer to ‘p‘. So, ‘∗p‘ and ‘∗∗pp‘ both give the value of first element of ‘arr‘, which is 7. Therefore, ‘∗p + ∗∗pp‘ results in ‘7 + 7‘, and the program prints ‘14‘.

Problem 2-2. Why BSTs?

In the last assignment you saw how a binary search tree was more efficient at solving the runway problem because inserting elements into a BST also meant implicitly sorting them. Prove that an in-order traversal of a binary search tree always produces the elements of the tree in sorted order.
Solution: In-order traversal follows the pattern of visiting left child (and its subtree), then visit the current node, and visit the right child (and its subtree).

The in-order traversal of a BST always produces elements of the tree in sorted order since by visiting the left child first, we are visiting the nodes with smaller keys first relatively. We are guaranteed to include the left node, the current node, and the right node for every subtree and by definition, left node $<$ current node $<$ right node for each subtree.
Therefore, an in-order traversal of a BST always produces the elements of the tree in sorted (ascending) order.

Problem 2-3. Linked Lists

(b) Unlike arrays, linked lists do not require you to store the length of the list, thus providing flexibility when adding elements. You are asked to print exactly half a linked by traversing the list exactly once, without knowing the length of the list before hand. Describe the algorithm you would use here and then code up the solution in the function printHalfList(). Print the output in the form Half List: a b c d as described earlier on a new line.
Solution: So, here we basically use the two-pointer technique to print the first half of a linked list.
Note: This idea was referenced from online articles like: https://www.geeksforgeeks.org/split-a-circular-linked-list-into-two-halves/
We first initialize ‘slowptr‘ and ‘fastptr‘, to the head of the list. Then, in each iteration of the loop, ‘slowptr‘ moves one step forward (i.e., to the next node), and ‘fastptr‘ moves two steps forward. When ‘fastptr‘ reaches the end of the list, ‘slowptr‘ will be at the midpoint of the list because it was moving at half the speed.
This algorithm gets us the half printed list because of the way we initialize the pointers.

(d) C requires you to manage memory usage on your own. Look up the free function that C provides and explain what its purpose is.
Solution: The ‘free‘ function in C is used to deallocate memory that was previously allocated by ‘malloc‘, ‘calloc‘, or ‘realloc‘. If we unnecessarily keep the memory allocated without use, it can pile up and cause the program to crash/hang. So, we use ’free’ to free the memory and prevent such memory leak.