Programming Assignment 6

Trees

(Deadline: Dec 8, 23:59)

Instructions for the output: The below instructions apply to all the programs (even for the upcoming labs).

While printing the output, there is no new line

1. between any two output lines
2. after the last output line
3. Before the first input line

Refer to Appendix 1 for more details.

If the program fails the sanity check, it should just print **“error”,** nothing less or more. Please note that “error” is different from “ERROR” and “Error” in C.

==

==

REGULAR EXERCISES

There are 3 regular exercises only (no bonus exercise). You need to make only one program for all the 3 exercises. Since the exercises differ in the number of input parameters, the first argument to the program should be the exercise number. Refer to the test cases for further clarification.

Sanity checks required:

1. The number of input parameters should be in accordance with the exercise number.
2. You can assume the input parameters to be integers. However, you should verify them for their sign (positive/negative).

==

Exercise 1. Write a C program to

a. create a binary tree using the given array
b. find the maximum element in the created tree. If there is more than 1 maximum element (with the same value), report the first occurence.
c. Report the level of the maximum element

   Note: Consider the level of the root node as 1.

Input format: ./a.out [exercise number] [Elements of the array]

Output format: [maximum element] [the level of the maximum element]

Test Case1: ./a.out 1 5 12 67 34 88 22 77 *(First argument is the exercise number)*

88 3

Test Case2: ./a.out 1 300 18 300 80 0 300

300 1

Test Case 3: ./a.out 1 1

1 1

Test Case 4: ./a.out 1

# *(no elements in the array)*

Test Case 5: ./a.out

error *(incorrect number of arguments)*

==

Exercise 2. Write a C program to create a binary tree using the given array and find the sum of the elements at the specified level.

Note: Consider the level of the root node as 1.

Input format: ./a.out [exercise number] [Elements of the array] [level number]

Output format: [sum of the elements at the specified level]

Test Case1: ./a.out 2 1 2 3 4 2 *(First argument is the exercise number.)*

5 *(Last argument is the level number.)*

Test Case2: ./a.out 2 3 18 200 200 80 1

3

Test Case 3: ./a.out 2 1 2 3 5 4 6 7 8 9 1 2 3 4 5 6 7 8 9 5

24

Test Case 4: ./a.out 2 1 4

# *(No element at level 4)*

Test Case 5: ./a.out

error *(incorrect number of arguments)*

==

Exercise 3. Create a binary tree using a linked list. The elements are given as input. Print the inorder traversal of the created tree.

Input format: ./a.out [exercise number] [elements of the linked list]

Output format: [elements in inorder traversal separated with space]

Test Case1:

```
./a.out 3 A B C D E F G H I
H D I B E A F C G
```

Test Case2: `./a.out 3 1 12 3 7 8 9 15 13 17`

```
13 7 17 12 8 1 9 3 15
```

Test Case 3: `./a.out 3 1 2 3`

```
2 1 3
```

==
==

SUBMISSION INSTRUCTIONS

Create a folder named [your_roll_number]_[your_name]_regular

[your_roll_number]_[your_name]_regular: This folder should have one file named [your_roll_number]_[your_name]_p1.c.

Zip the folder as [your_roll_number]_[your_name]_lab6. Upload the zip file to the Google classroom. Kindly use your complete roll number as assigned by the institute.

==

==

Appendix 1

Your program should not print any unnecessary new lines (or even tabs and spaces). As an example, assume that your code is printing 2 numbers as output, n (having value 10) and m (having value 40). Then your sequence of printf statements should be

```
=====================================
printf("%d",n);
printf("\n");
printf("%d",m);
```

This will result in an output like

```
10
40
=====================================
```

and not any of the following

1.

```
printf("%d",n);
printf("\n\n");
printf("%d",m);
```

Here, there is an additional newline between two output lines.
This will result in an output like

```
10

40
=====================================
```

2.

```
printf("%d",n);
printf("\n");
printf("%d",m);
printf("\n");
```

Here, there is an additional newline after the last output line.
This will result in an output like

```
10
40

=====================================
```

3.

```
printf("\n");
printf("%d",n);
printf("\n");
printf("%d",m);
```

Here, there is an additional newline before the first output line.
This will result in an output like

```

10
40
======================================
```