Lab 3 CS201 (2020)

Read the assignment carefully. Only 1 program to be submitted.

- Single .c file. Program must be compilable using gcc as we will be likely using linux platform to evaluate your assignments
- You MUST follow the Filename format :
    - L03_<Your First name>_<Entrynumber>_CS201_2020.c
        - For e.g. L03_Raman_2019CSB9999_CS201_2020.c
- Input outputs that you tried or used for testing your code can be included in your program file at end within comments (using /* */) and/or in google
- If you know you code is not working for some cases etc, do mention that clearly in comments section at the beginning of your code (using /* ….. */) and also in google form
- There shall not be any identifiable information within the program. No reference to your name, roll number etc. shall appear in your code.
- Deadline as of now:
    - 7th Oct 2020, 11:59:59 PM
    - While submitting, exercise due care and caution to submit the appropriate code (your last updated program, not your intermediate trial codes) and with proper filename.
    - Use only Google form to submit the assignments. Assignment submitted via other means (Google Classroom / Moodle / Email) won't be considered for evaluation unless some prior approval taken for some emergency genuine reasons.
- No Plagiarism Please. Please note that we may do plag check now or later even after endsem, and you will be then (severely) penalized. You must do your own coding and neither take nor provide codes from/to anyone else.
    - Those who complete their works, they are kindly requested to not share their codes in name of friendship or so. You may get severely penalized for the same and by sharing codes, you are not really helping them either.
    - We sincerely request and hope that you will try your level best and not adopt any unfair means so that we need NOT go through the painful experience of penalizing you.

==========Assignment (To be implement in C using pointers etc) ==========

Write the program in C to practice the BST/AVL-tree/Binary tree concepts.

/* Refer below to some query types with their functionality explained..
First character in each row would be corresponding to query type and then it is followed by some parameters (integers) as and when applicable

Assume – all elements values in the trees are positive and distinct.
If you get a query to add/insert a duplicate / non-positive value, don't insert

Tree info would be only accessible only through single pointer i.e. pointer to root node.
*/

Query Type / Identifier	Parameter / Values following the Query type	Functionality / Response / Remarks

T	//Sequence of Integers
	
	a1 a2 a3 -a2 -a4 a5 .... an
	
	//Assume all ai are +ve integers
		// Any valid 8 bit integer
		
		Construct BST by adding first a1, then a2, then a3, removing a2, removing a4, adding a5 ....
		
		// Need not insert duplicate value (a5==a3)
		// If a4 does not exist, don't do anything
		// Need not print anything

H	//Sequence of Integers
	
	a1 a2 a3 -a2 -a4 a5 .... an
	
	//Assume all ai are +ve integers
		// Any valid 8 bit integer
		
		Construct AVL by adding first a1, then a2, then a3, removing a2, removing a4, adding a5 ....
		
		// Need not insert duplicate value (a5==a3)
		// If a4 does not exist, don't do anything
		// Need not print anything

A	i j .. k .. //Seq. of integers	Add nodes with values i j k in the tree in this sequence I j k //Consider last constructed tree. If no tree, then consider simple BST that was initially empty). If any i/j/k is 0 or negative, do nothing.

U	i j .. k ..
	
	//Seq. of integers
		Uproot or say Delete the nodes if they are there in the tree (last constructed tree).
		// If any i/j/k is o or negative, do nothing
		//Whenever delete requires replacement, use left subtree if there is choice (left vs right)

Query	Parameter /	Functionality / Response / Remarks

Type / Identifier	Values following the Query type (integer values)	
F	x	Find if x there in the tree. Print "Yes" or "No" accordingly. Ofcourse it would be No if x<0
Q		Output no. of leaves in the tree
N		Output no. of nodes in the tree
P		Print Preorder traversal of tree . Use non recursive code
I		Print Inorder traversal (recursive code)
S		Print Postorder traversal (recursive code)
L		Print the level order traversal of tree
D		Print Depth/Height of the tree (Height of single node tree is 0)
W		Width of the tree (I will explain its definition in the class)
C	i j	Lowest common ancestor of node i and node j. If node i or node j does not exist, print -1. if i==j print i.
R	i j	Print the Route/Path from (ancestor) node i to (descendant) node j if there is any (E.g. print: i L t1 R j). If node i does not exist, print "Source does not exist" else if j does not exist, print "Destination does not exist". If no path to go from i to j, print "Unreachable". If i==j, it would print: i
X		Print the length of the Diameter of the tree (I will explain its definition in the class)
Y		Perimeter of the tree. Space separated boundary traversal (clockwise) beginning from root (I will explain more in the class)

Some Assumptions/Constraints -

1. Left sub-tree of a node has values less than or equal to node value. Values in right sub-tree are strictly greater.

2. Elements in the tree are positive. (0 and negative elements not to be there in tree). If you encounter 0 in query type : T H A or U do nothing.

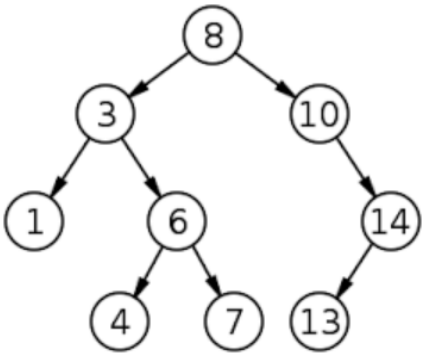
3. At anytime , there is only one tree in the system. Initially assume BST and its empty i.e. root=NULL. Whenever you encounter query type T or H, you delete the previous tree and construct the new tree as per the query. Subsequent queries will then to be performed on this last constructed tree. When you delete the tree, free the space. Don't just make root=NULL.

4. First line of the input would be Z i.e. number of queries and then there would Z lines - each line indicating a query. For query types T H A and U there would be no output (your code works internally in constructing/modifying the tree as per query.). For other queries there would be single line output

5. Query A and U operates on previously constructed tree. If that was simple BST, then add/delete nodes preserving BST property. If most recently constructed tree was AVL ,

then add/delete nodes preserving AVL tree property. By default, assume simple BST (This case would only happen if you observe query type A but you had not observed T or H so far). Similarly for U as and when applicable.

Consider this tree **(Tree 1)**



This BST tree would be constructed if input query is

T 8 3 1 10 14 13 6 5 7 -5 4
or
T 8 3 1 6 4 7 10 14 13

Consider only the left subtree (rooted at 3). Thats AVL tree in itself. It can be constructed if input query is
H 3 1 4 6 7
or
H 1 3 6 4 7

Sample Input Output Queries for following BST tree **(Tree 2)** of three elements 3 ,2 and 1

```
    3
   /
  2
 /
1
```

Input Queries	**Outputs**
R 2 2	2
R 2 1	2 L 1
R 3 2	3 L 2
R 3 1	3 L 2 L1
R 2 4	Destination does not exist
R 0 2	Source does not exist
R 2 3	Unreachable

Some Sample Input Output Queries for an AVL tree **(Tree 3)**

ii)

Input Queries	**Outputs**
R 2 3	2 R 3
R 1 3	Unreachable
R 1 2	Unreachable
R 2 1	2 L 1

==========

Some example Queries answers:

A 5
A 7
A -2
A 3 6 9 -3 7 -11 25 78 96

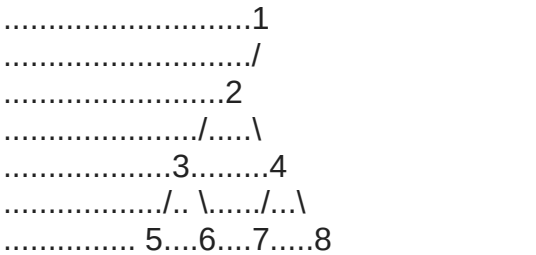
THESE WOULD PERFORM FOLLOWING
ADD NODE WITH VALUE 5
ADD NODE WITH VALUE 7
DO NOTHING
ADD NODES WITH VALUES 3 6 9 7 25 78 96 ONE BY ONE

UPROOT /DELETE THE NODE WILL ONLY DELETE THAT NODE AND NOT THE TREE UNDERNEATH

0 VALUES ==> DO NOTHING

Doubt: In perimeter traversal we have to print the sum only or the whole path values too ? ==>
perimeter = sum of the value of boundary tree traversal elements

Example consider the following tree **(Tree 4).**



Clockwise Boundary Traversal is : 1 8 7 6 5 3 2 & Anti-clockwise is : 1 2 3 5 6 7 8

Perimeter is 32

So if the query is as follows on above tree, then the output would be 32 as mentioned here .

Sample Input:	**Sample Output**
Y	32 1 8 7 6 5 3 2

//For Trees 1, 2, and 3 mentioned above, the perimeter is 60, 6 and 6, respectively

Sample Inputs	**Sample Outputs**
7 T 10 12 9 6 11 15 U 6 I L Y X D	9 10 11 12 15 10 9 12 11 15 57 10 12 15 11 9 4 2
13 H 10 1 8 7 -10 A 4 5 U 8 P I S L X Y D W C 1 5 R 1 7	4 1 7 5 1 4 5 7 1 5 7 4 4 1 7 5 4 17 4 7 5 1 2 2 4 Unreachable

Also, note if you have imbalanced AVL after some insertion/delete and there are multiple options applicable e.g. both left-left case and left-right case applicable, then use the one with least rotations, so use Left-Left compared to Left-Right (and similarly, use Right-Right rather than Right-Left, in cases when both choices are possible)