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

T a1 a2 a3 -a2 -a4 a5 .... an //Construct BST by adding first a1, then a2, then a3, removing a2, removing a4, adding a5 ....
H a1 a2 a3 -a2 -a4 a5 .... an //Construct height balanced tree i.e. AVL by adding first a1, then a2, then a3, removing a2, removing a4, adding a5 ....
A i j .. k //Add nodes i j k in the tree. If any i/j/k is negative, do nothing
F x //Find if x there in the tree. Print "Yes" or "No" accordingly. Ofcourse it would be No if x<0
U i j .. k //Uproot or say Delete the nodes if they are there in the tree. If any i/j/k is negative, do nothing

L //No of leaves in the tree
N //No. of nodes in the tree

P //Preorder
I //Inorder
S //PostOrder
L //Print the tree level order traversal of tree

D //Depth/Height of the tree
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 node i to node j if there is any (E.g. print: i Left i2 Right i3 Reached 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 reached j
X //Print the length of the Diameter of the tree
Y //Perimeter of the tree. SPace separated, clockwise beginning from root.

E.g. following 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 below. 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

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. Preorder traversal to be implemented without recursion. For other parts, you can choose recursion if you so want.
4. At anytime , there is only one tree in the system. Initially 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 constructed tree. When you delete the tree, free the space. Dont just make root=NULL.

5. First line of the input would be Q i.e. number of queries and then there would Q 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

6. 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.

More input output examples would be shared soon either via email or in class. Feel fee to let me know if you have any questions.

I recommend you to write code yourself and not depend on internet or others. As that would likely lead to plagiarism for which I have to respond in a way that the concerned person may not be able to appreciate because of his/her limited thinking.