Informatics II
Exercise 8

Learning Goals:

- Practise inserting items and deleting items in Binary Search Tree
- Implement functions of inserting items, deleting items, and searching elements in Binary Search Tree in C
- Implement Binary Search Trees in C.
- Validate implementation of Binary Search Trees.

Binary Search Trees

Task 1. Binary search tree is created by inserting nodes 12, 21, 15, 32, 13, 3, 8, 5, 1, 42, 17. Show structure of the tree after insertion of each node.

Initial tree
Empty binary search tree.

Task 2. Considering binary search tree created in `Task 1`, show the structure of the tree after deleting each of the following nodes 5, 32, 15, 21, 12. When deleting the node has two children, we first find the item with the largest value in the left subtree.

Task 3. The structure of the tree node is defined as follow. The entry to a binary search tree is the root that is also a tree node.

```
1  struct TreeNode {
2    int val;
3    struct TreeNode* left;
4    struct TreeNode* right;
5  };
```

Write a C program that contains the following functions:

a) Write the function *void insert(struct TreeNode** root, int val)* that inserts an integer `val` into the binary search tree. Note that, you need to create a tree node for `val`, and find the correct position to insert the new tree node.

b) Write the function *struct TreeNode* search(struct TreeNode* root, int val)* that finds the tree node with value `val` and returns the node. If not exist, return NULL;

c) Write the function *void delete(struct TreeNode** root, int val)* that deletes the node with value `val` from the tree.

d) Write *void printTree(tree *root)* which prints all edges of the tree from root in the console in the format `Node A -- Node B`, and each edge is printed in a separate line. The ordering of the printed edges does not matter and may vary based on your implementation.

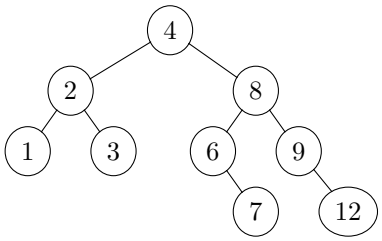
e) Write *struct TreeNode* maximum(struct TreeNode* node)*, which returns the node with the largest value in the subtree with root node `n`.

f) Write *struct TreeNode* minimum(struct TreeNode* node)*, which returns the node with the smallest value in the subtree with root node `n`.

g) Write *int distanceToRoot(struct TreeNode* root, int val)* that returns the distance of the node with value `val` from the root node `root`. Assume that val exists in the Tree.

- Test your above implementations. For example, assume that the values 4, 2, 3, 8, 6, 7, 9, 12 and 1 are inserted into an empty tree, your program should produce the binary tree shown below.

Binary Tree Form



Test your program by performing the following operations:

- Create a root node `root` and insert the values 4, 2, 3, 8, 6, 7, 9, 12, 1.
- Print tree to the console.
- Print the minimum value of the tree.
- Print the distanceToRoot of node 7.
- Delete the values 4, 12, 2 from the tree.
- Print tree to the console.
- Print the maximum value of the tree.
- Print the distanceToRoot of node 6.

Task 4. Given a rooted tree T, the *lowest common ancestor (LCA)* of two nodes *n1* and *n2* is defined as the node that is farthest from the root in T and has both *n1* and *n2* as descendants (where we allow a node to be a descendant of itself). Consequently, the LCA of *n1* and *n2* in T is the shared ancestor of *n1* and *n2* that is located farthest from the root. Note that *n1* and *n2* are in the tree T.

For example, given a tree as in Figure 1, the LCA of nodes *9* and *21* is node *15*.

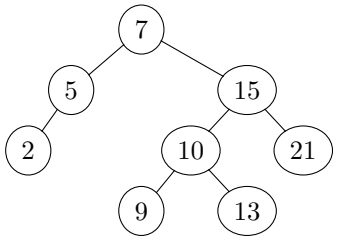


Figure 1: LCA of nodes 9 and 21 is node 15

Given values of two nodes in a Binary Search Tree, implement in C the function `struct TreeNode* lca(TreeNode* root, int n1, int n2)` that finds the *Lowest Common Ancestor (LCA)*. You may assume that both the values exist in the tree.

Write in C a program to test your implementation by performing the following operations:

- Create an empty tree and insert the nodes 7, 5, 6, 1, 9, 10, 8.
- Print the resulting tree using method described in **Task 3**.
- Print out the LCA of node *8* and node *9*