National University of Computer and Emerging Sciences

Lab Manual

for

Data Structure

Course Instructor	Mr. Razi Uddin
Lab Instructor(s)	Ms. Mamoona Akbar Ms. Mateen Fatima
Section	DS (BSE-3B)
Semester	FALL 2023

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual 08

Objectives:

After performing this lab, students shall be able to Practice:

- ✓ Iterator related to link list
- ✓ Implementation of Binary Search Tree

Task 1

Consider a linked list that stores integers. Write a C++ program that performs the following tasks using iterators:

1. Create a linked list with 5 nodes, initially containing the integers 10, 20, 30, 40, and 50.

2. Display the elements of the linked list using an iterator.

3. Add a new node with the value 60 at the end of the linked list.

4. Display the updated elements of the linked list using an iterator.

5. Remove the node containing the value 30 from the linked list.

6. Display the final elements of the linked list using an iterator.

Note: Use iterators to traverse and manipulate the linked list.

Output:

```
Original List:
10 20 30 40 50

List after adding 60:
10 20 30 40 50 60

List after removing 30:
10 20 40 50 60
```

Task 2

Implement the following Tree Node:

```
struct Node
{
        int  data;
        Node*left;
        Node *right;
};
```

Now implement a binary search tree class “BST” which contains the root of type **Node** as a data member.

```
class BST
{
        Node* root;
};
```

Implement the following member functions for your binary search tree:

NOTE: Use helper functions if required.

1. A default Constructor which sets the root to nullptr.
2. Implement a function ‘insert’. It should insert the data while considering the insertion rules. If the data already exists in the BST, simply return false and true otherwise.
   `bool insert(int v)`

3. A copy constructor which uses recursion to deep copy another Binary Search Tree object.

4. A function “inorderPrint” prints the keys using in-order traversal.
5. `void inorderPrint () const`
6. Use level order traversal for the printing of trees, level by level.
   `void levelorderPrint () const`
7. A function “search”. The function then uses recursion to return a pointer to the corresponding node. If the key does not exist, the function returns nullptr.
   `Node* search(int key)`
8. Use inorder LVR to implement a recursive function “countNodes” to return the count of total nodes in BST.
9. `int countNodes() const`
10. Use Preorder traversal VLR to implement a recursive function “leafCount” to return the count of leaf nodes in BST.
11. `int leafCount() const`
12. Use Postorder LRV to implement the Destructor for BST.

Task 3

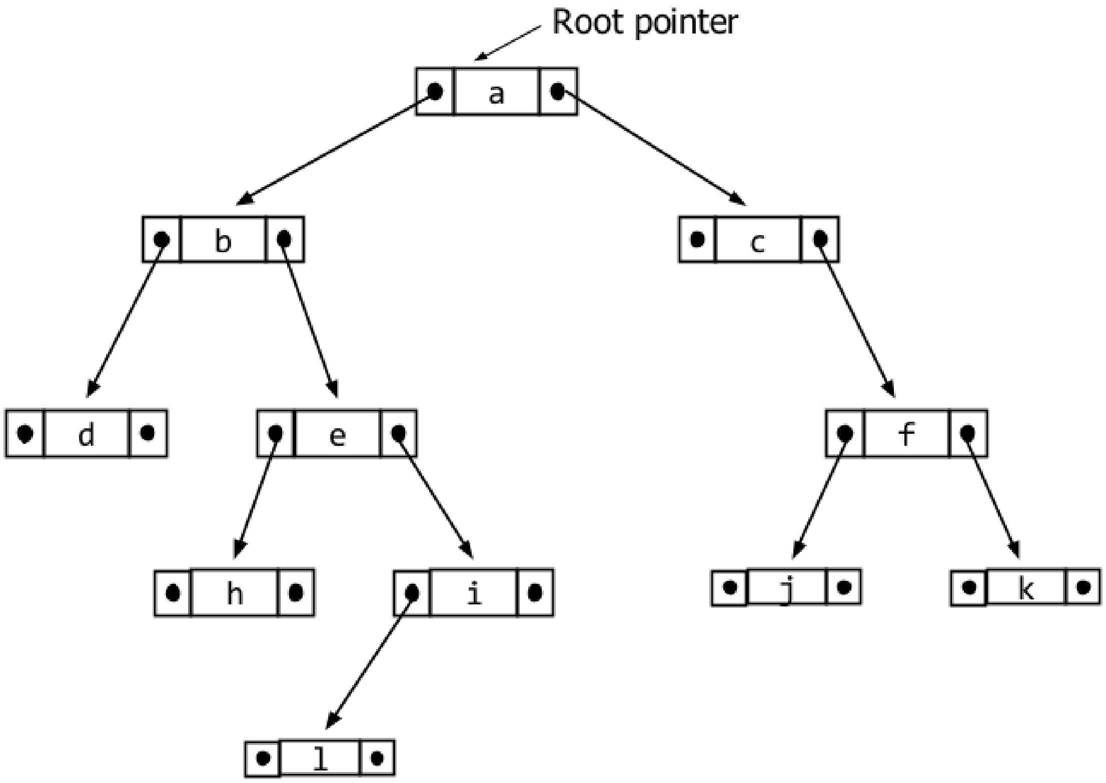
Consider the Link List based tree implementation and traverse the tree in depth-first order, your code should print second largest depth of tree and print leaf Nodes only.

```
void testFunction(Tree* root){
//code here
//print Tree leafs Nodes inside this function
//by calling printLeafs(Tree* root);

//Print Second Largest depth of Tree inside this function
//by calling int SecondLargestdepth (Tree* root);
}
void printLeaf(Tree* root){
//code here
//print Tree leafs Nodes inside this function
}
int SecondLargestdepth (Tree* root){
//code here
//Print Second Largest depth of Tree inside this function
//Return Second Largest depth of Tree
}
```

Output

```
Binary tree leaf nodes are
D H L J K
Second Largest depth of Tree is 3.
```


Root pointer
a
b
c
d
e
f
h
i
j
k
l