Name : Mufaddal Diwan
Class : SE-3
Roll no : 21340

ASSIGNMENT NO. 1

TITLE	Binary Tree Implementation, traversals and operations
PROBLEM STATEMENT	Create binary tree with n nodes, perform following operations on it: • Perform inorder/preorder and post order traversal • Create a mirror image of it • Find the height of tree • Copy this tree to another [operator=] • Count number of leaves, number of internal nodes. • Erase all nodes in a binary tree. (implement both recursive and non-recursive methods)
OBJECTIVE	To understand construction of binary tree and its traversal techniques.
OUTCOME	At the end of this assignment students will able to construct a Binary tree and perform basic operations on Binary tree.
S/W PACKAGES AND HARDWARE APPARATUS USED	1. (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS **2.** Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK

Concepts related Theory:

Binary tree is specific type of tree in which each node can have atmost(zero,one,two) two children namely left child and right child. Empty tree also a valid binary tree.

In computer science**,** tree traversal is a form of graph traversal and refers to the process of visiting each node in a tree data structure, exactly once. Such traversals are classified by the order in which the nodes are visited.

Data structures for tree traversal:

Traversing a tree involves iterating over all nodes in some manner. Because from a given node there is more than one possible next node then, assuming sequential computation, some nodes must be deferred—stored in some way for later visiting. This is often done via a stack (LIFO) or queue (FIFO). As a tree is a self-referential (recursively defined) data structure, traversal can be defined by recursion

Depth-first search is easily implemented via a stack, including recursively, while breadth-first search is easily implemented via a queue, including corecursively.

Depth-first search:

These searches are referred to as *depth-first search* (DFS), as the search tree is deepened as much as possible on each child before going to the next sibling. For a binary tree, they are defined as display operations recursively at each node, starting with the root.

***Operations of binary tree*:**

- Traversal
- Creation
- Deletion
- Compare
- Merge

Traversing: Traversal refers to the process of visiting all the nodes of binary tree once. There are three ways for traversing binary tree.

1.*Pre-order:*

- Check if the current node is empty /null
- Display the data part of the root (or current node).
- Traverse the left subtree by recursively calling the pre-order function.
- Traverse the right subtree by recursively calling the pre-order function.

2.*In-order:*

- Check if the current node is empty/null
- Traverse the left subtree by recursively calling the in-order function.
- Display the data part of the root (or current node).
- Traverse the right subtree by recursively calling the in-order function.

3.*Post-order:*

- Check if the current node is empty/null
- Traverse the left subtree by recursively calling the post-order function.
- Traverse the right subtree by recursively calling the post-order function.
- Display the data part of the root (or current node).

Algorithm:

ALGORITHM INORDERTRAVERSE()

{

1. set top=0, stack[top]=NULL, ptr = root
2. Repeat while ptr!=NULL
    2.1 set top=top+1
    2.2 set stack[top]=ptr
    2.3 set ptr=ptr->left
3. Set ptr=stack[top], top=top-1
4. Repeat while ptr!=NULL
    4.1 print ptr->info
    4.2 if ptr->right!=NULL then
        4.2.1set ptr=ptr->right
        4.2.2 goto step 2
    4.3 Set ptr=stack[top], top=top-1

}

ALGORITHM PREORDERTRAVERSE()

{

1. set top=0, stack[top]=NULL, ptr = root
2. Repeat while ptr!=NULL
    2.1 print ptr -> info
    2.2 if (ptr -> right != NULL)
        2.2.1 top = top +1
        2.2.2 set stack [top] = ptr -> right
    2.3 if (ptr -> left != NULL)
        2.3.1 ptr=ptr -> left

else

    2.3.1 ptr=stack[top], top=top-1

}

ALGORITHM POSTORDERTRAVERSE()

{

1. set top = 0, stack [top] = NULL, ptr = root
2. Repeat while ptr!=NULL
    2.1 top = top +1 , stack [top] = ptr
    2.2 if (ptr -> right != NULL)
        2.2.1 top = top +1
        2.2.2 set stack [top] = - (ptr -> right)
    2.3 ptr = ptr -> left
3. ptr = stack [top], top = top-1
4. Repeat while (ptr > 0)
    4.1 print ptr -> info
    4.2 ptr = stack [top], top = top-1
5. if (ptr < 0)
    5.1 set ptr = - ptr
    5.2 Go to step 2

}

Test-Cases

Description	Input	Output	Result
Create Tree (Enter -1 if no node)	6 5 7 -1 8 2-1 -1 -1 -1 -1	-	Pass
Preorder traversal	-	6 5 7 8 2	Pass
Postorder traversal	-	2 8 7 5 6	Pass
Inorder Traversal	-	7 2 8 5 6	Pass
Height of tree	-	5	Pass

Conclusion: After successfully completing this assignment, Students will be able create an Expression tree and performs various operations on Binary tree