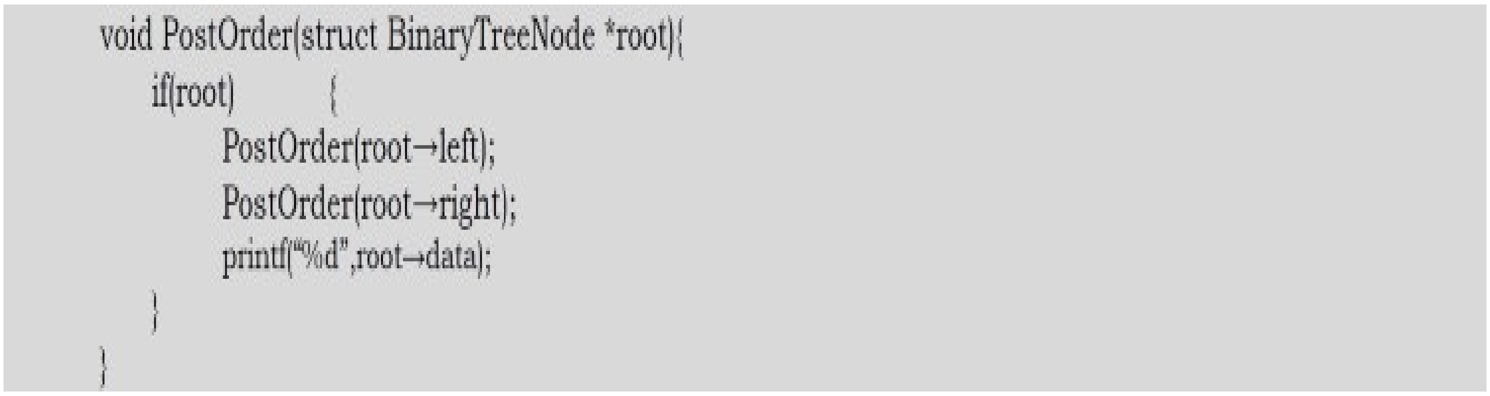
PostOrder Traversal

In postorder traversal, the root is visited after both subtrees. Postorder traversal is defined as follows:

- Traverse the left subtree in Postorder.
- Traverse the right subtree in Postorder.
- Visit the root.

The nodes of the tree would be visited in the order: 4 5 2 6 7 3 1



```
void PostOrder(struct BinaryTreeNode *root){
    if(root)        {
        PostOrder(root→left);
        PostOrder(root→right);
        printf("%d",root→data);
    }
}
```

Time Complexity: O(*n*). Space Complexity: O(*n*).

Non-Recursive Postorder Traversal

In preorder and inorder traversals, after popping the stack element we do not need to visit the same vertex again. But in postorder traversal, each node is visited twice. That means, after processing the left subtree we will visit the current node and after processing the right subtree we will visit the same current node. But we should be processing the node during the second visit. Here the problem is how to differentiate whether we are returning from the left subtree or the right subtree.

We use a *previous* variable to keep track of the earlier traversed node. Let's assume *current* is the current node that is on top of the stack. When *previous* is *current's* parent, we are traversing down the tree. In this case, we try to traverse to *current's* left child if available (i.e., push left child to the stack). If it is not available, we look at *current's* right child. If both left and right child do not exist (ie, *current* is a leaf node), we print *current's* value and pop it off the stack.

If prev is *current's* left child, we are traversing up the tree from the left. We look at *current's* right child. If it is available, then traverse down the right child (i.e., push right child to the stack); otherwise print *current's* value and pop it off the stack. If *previous* is *current's* right child, we are traversing up the tree from the right. In this case, we print *current's* value and pop it off the stack.