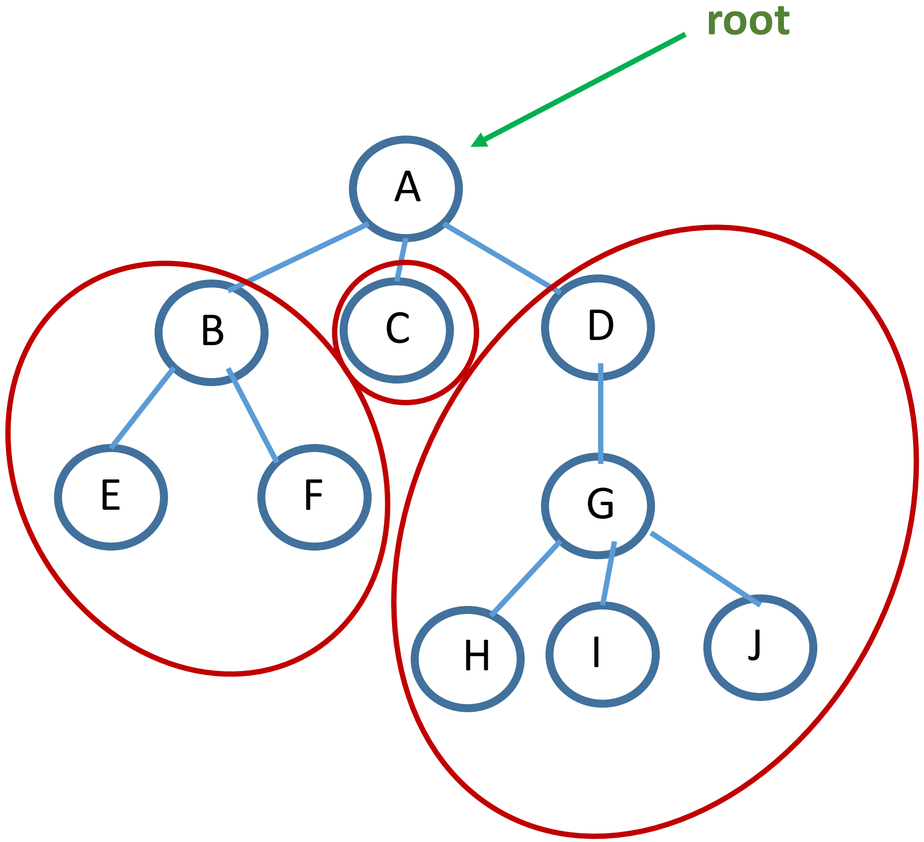
COMP 2611, Data Structures

LECTURE 6: TREES AND BINARY TREES

What is a Tree?

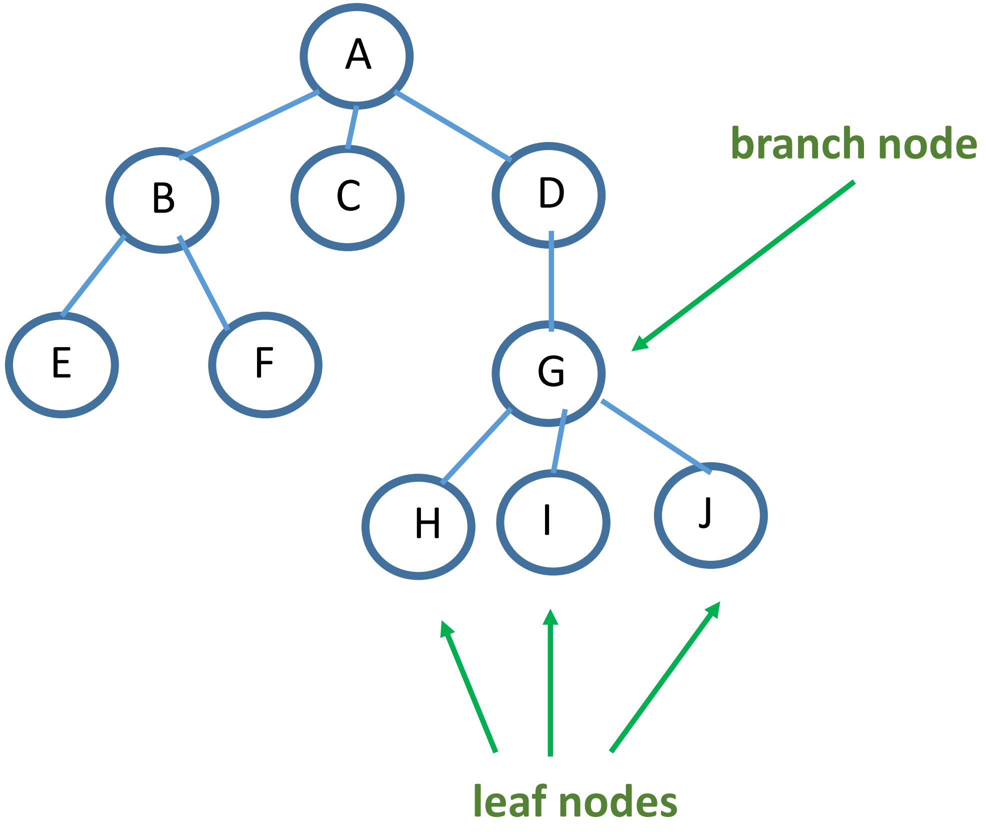
- A tree is a finite set of nodes such that:
  - There is one specially designated node called the *root* of the tree.
  - The remaining nodes are partitioned into m ≥ disjoints sets T_1, T_2, …, T_m, and each of these sets is a tree.



- The root of the given tree is A.
  - There are three subtrees rooted at A.
  - The *degree* of a node is the number of subtrees of the node.

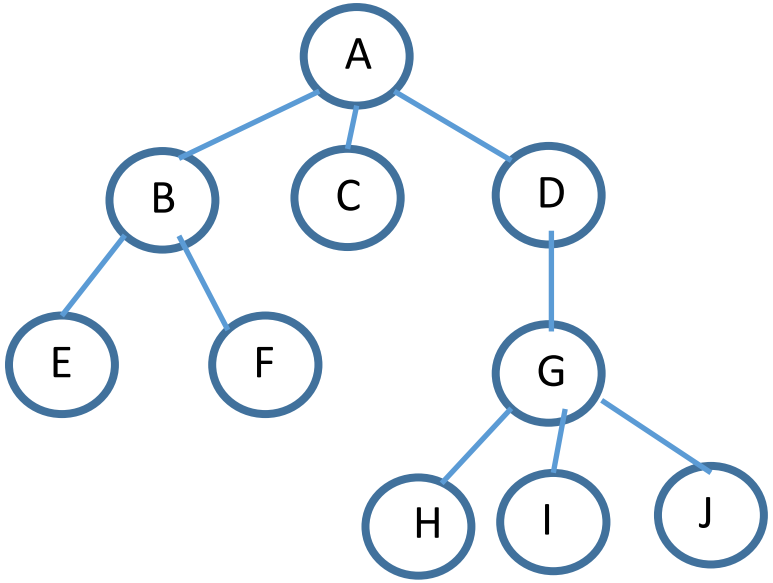
Tree Terminology

- The terms *parent*, *child*, and *sibling* are used to refer to the nodes of a tree.
- A node may have several children but only one parent (except for the root). The root is the only node that does not have a parent.
- *Sibling* nodes are child nodes of the same parent (e.g., *B*, *C*, *D*).
- A *terminal* node (or *leaf* is a node of degree 0). A *branch* node is a nonterminal node.



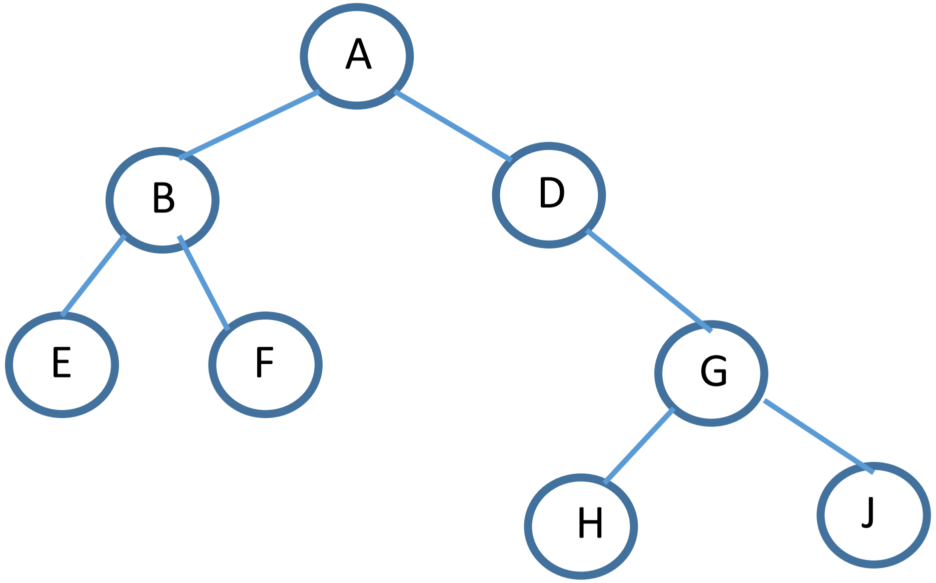
Tree Terminology

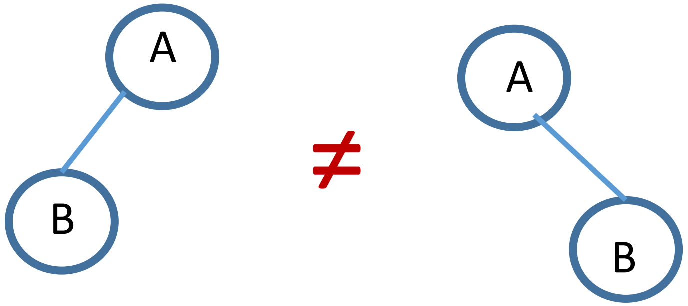
- The *moment* of a tree is the number of nodes in the tree.
- The *weight* of a tree is the number of leaves in the tree.
- The *level* (or *depth*) of a node is the number of branches that must be traversed on the path to the node from the root. The root has level 0.
- The *height* of a tree is the longest path from the root node to any leaf node in the tree.



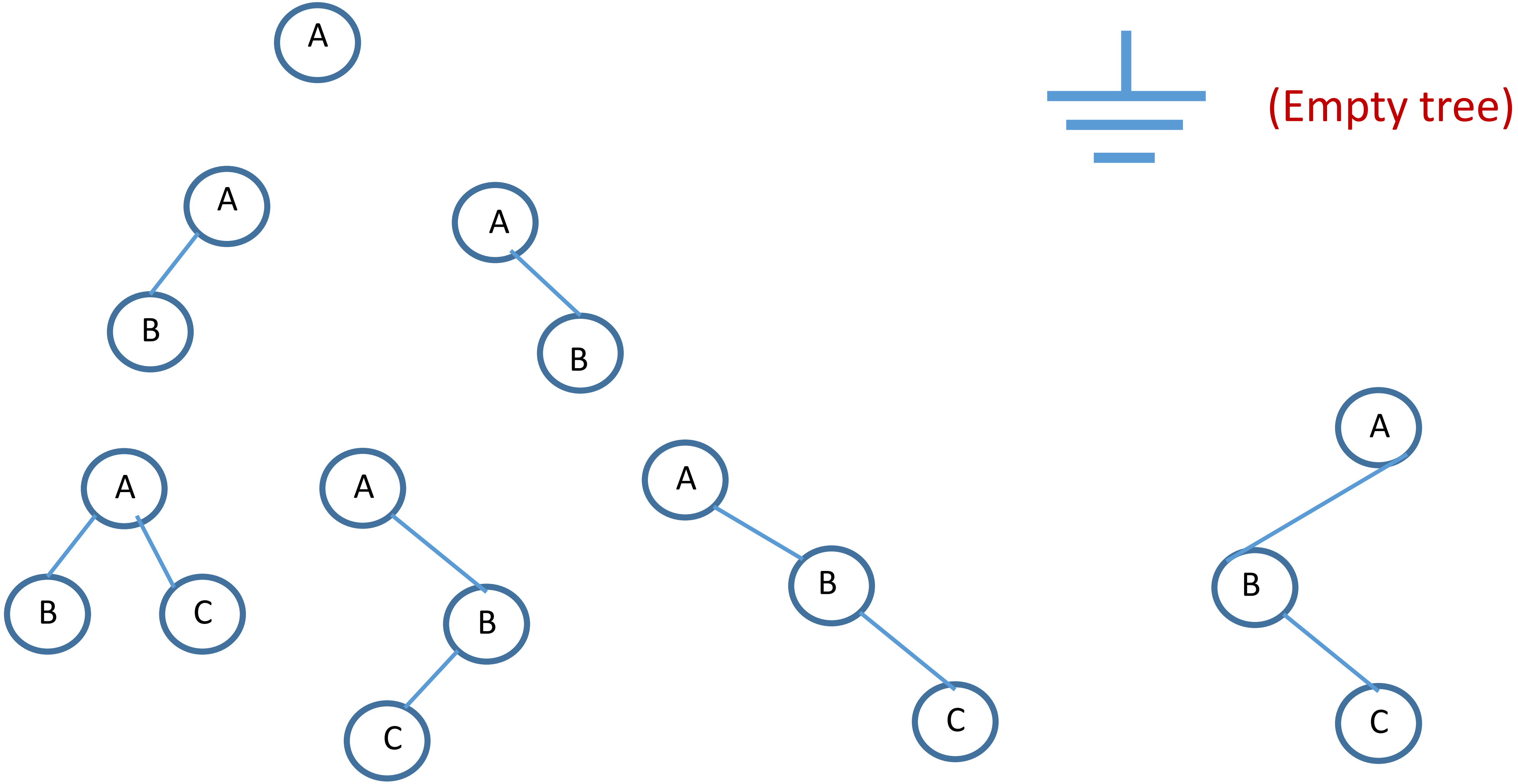
Binary Trees

- A binary tree:
  - Is empty, or.
  - Consists of a root and two subtrees– a left and a right– with each subtree being a binary tree.
  - If a node has one non-empty subtree it is important to distinguish whether it is on the left or right.





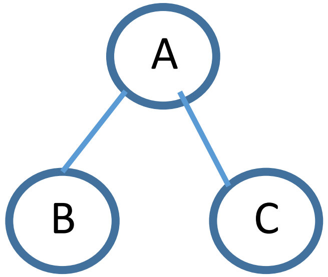
Examples of Binary Trees



(Empty tree)

Traversing a Binary Tree

Six possibilities:

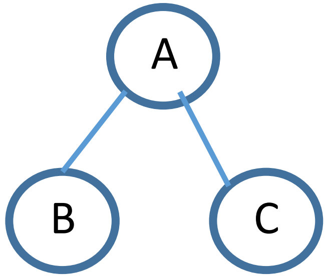


Root -> Left -> Right (ABC)
Root -> Right -> Left (ACB)

Left -> Root -> Right (BAC)
Right -> Root -> Left (CAB)

Left -> Right -> Root (BCA)
Right -> Left -> Root (CBA)

Traversing a Binary Tree



Root -> Left -> Right
~~Root -> Right -> Left~~

Preorder: A B C

Left -> Root -> Right
~~Right -> Root -> Left~~

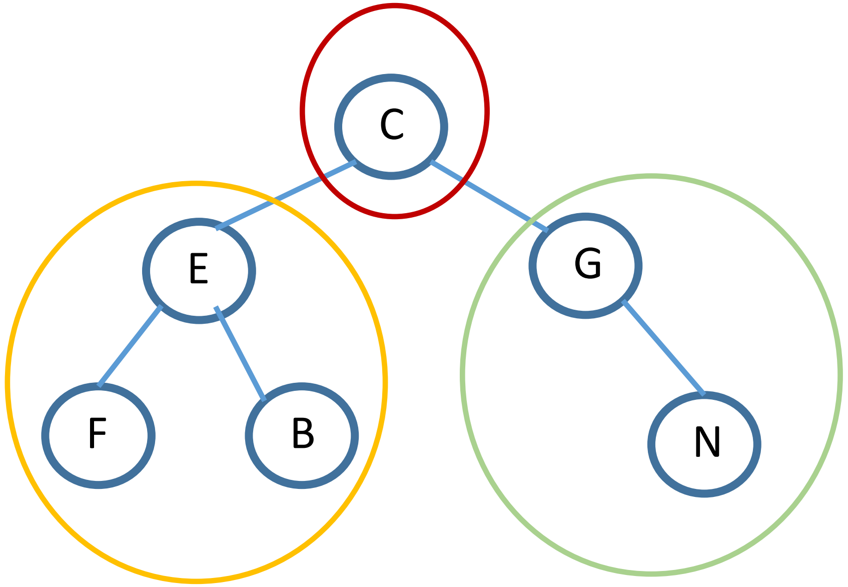
Inorder: B A C

Left -> Right -> Root
~~Right -> Left -> Root~~

Postorder: B C A

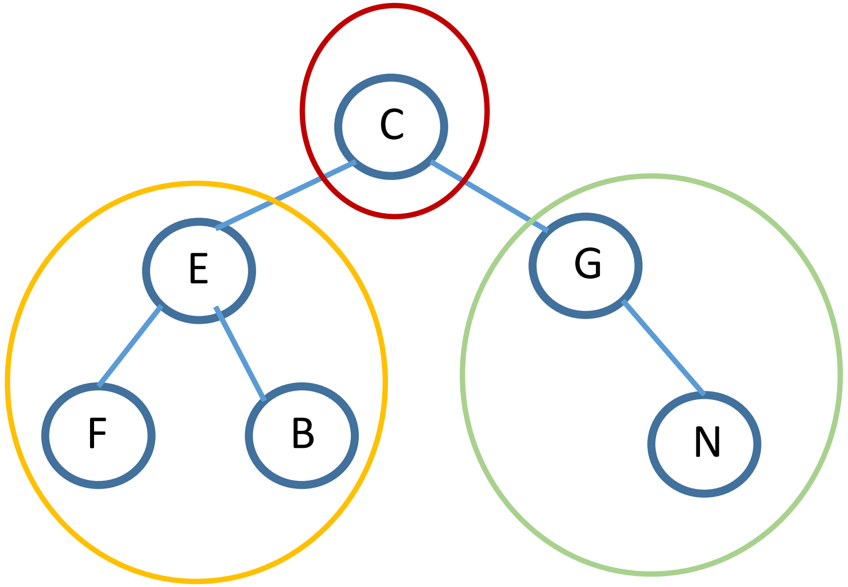
Note: Must "start" from root, A

Give the Preorder Traversal of This Binary Tree:



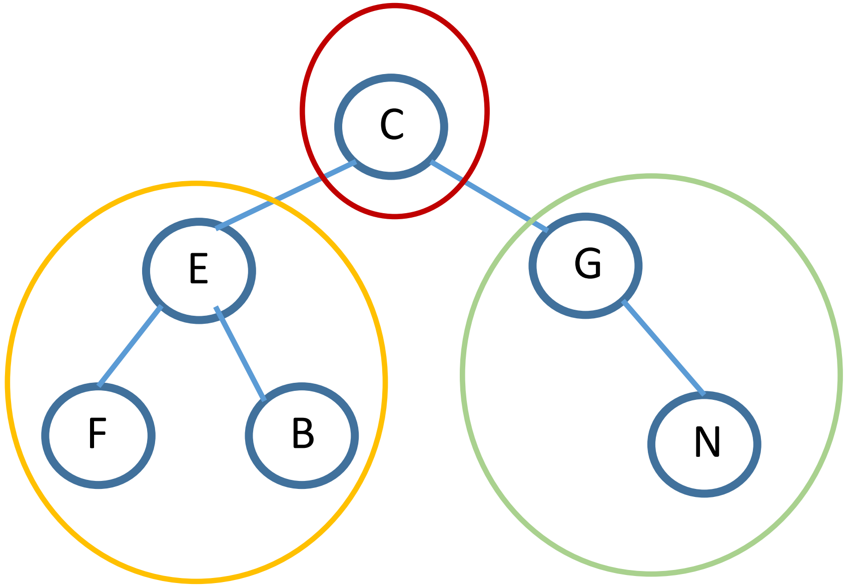
C E F B G N

Give the Inorder Traversal of This Binary Tree:



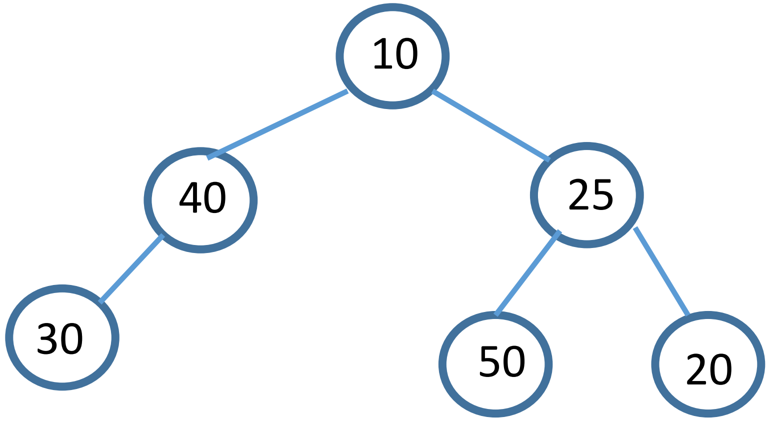
F E B C G N

Give the Postorder Traversal of This Binary Tree:

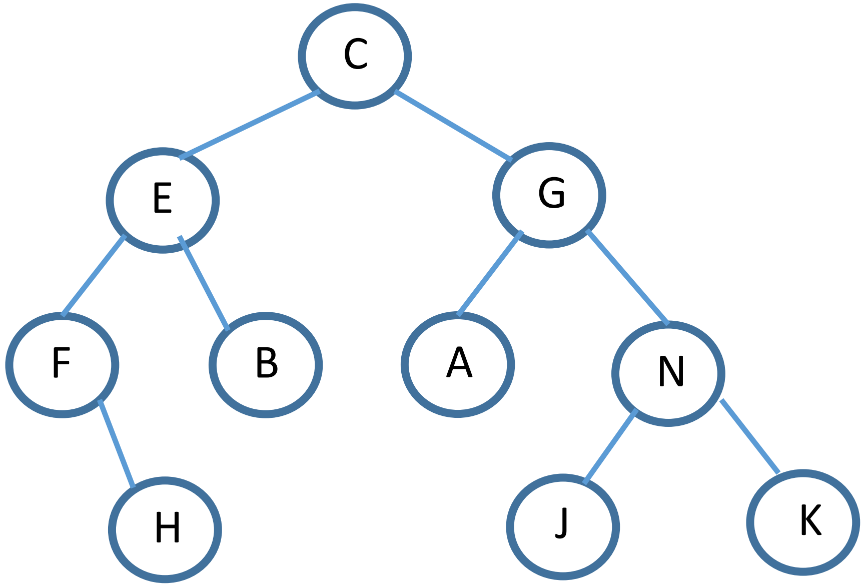


F B E N G C

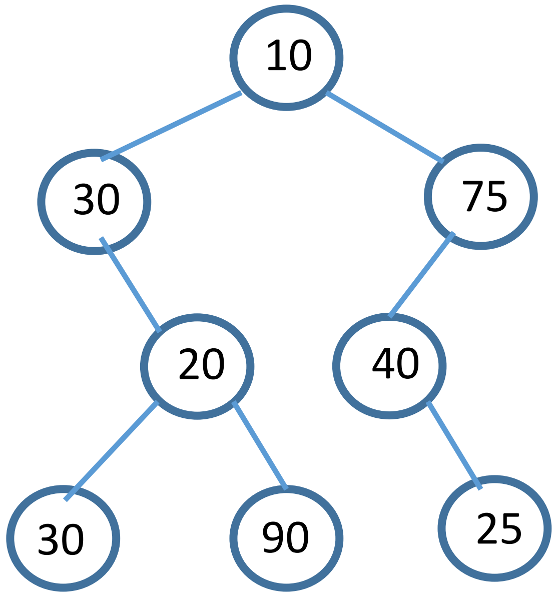
Give the Preorder, Inorder, and Postorder Traversals of This Binary Tree:



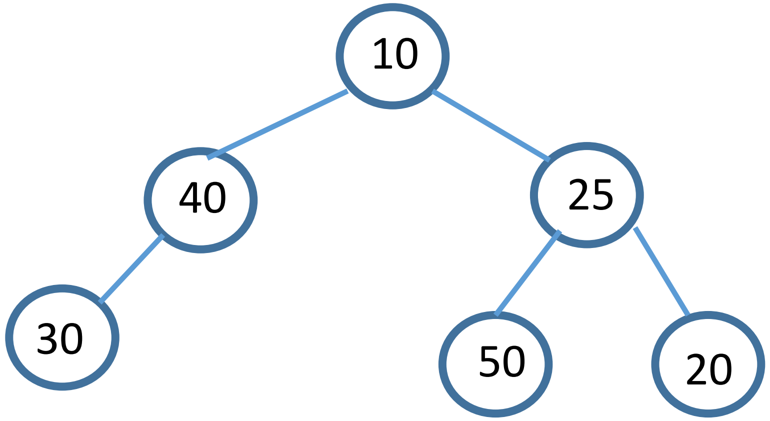
Give the Preorder, Inorder, and Postorder Traversals of This Binary Tree:



Give the Preorder, Inorder, and Postorder Traversals of This Binary Tree:



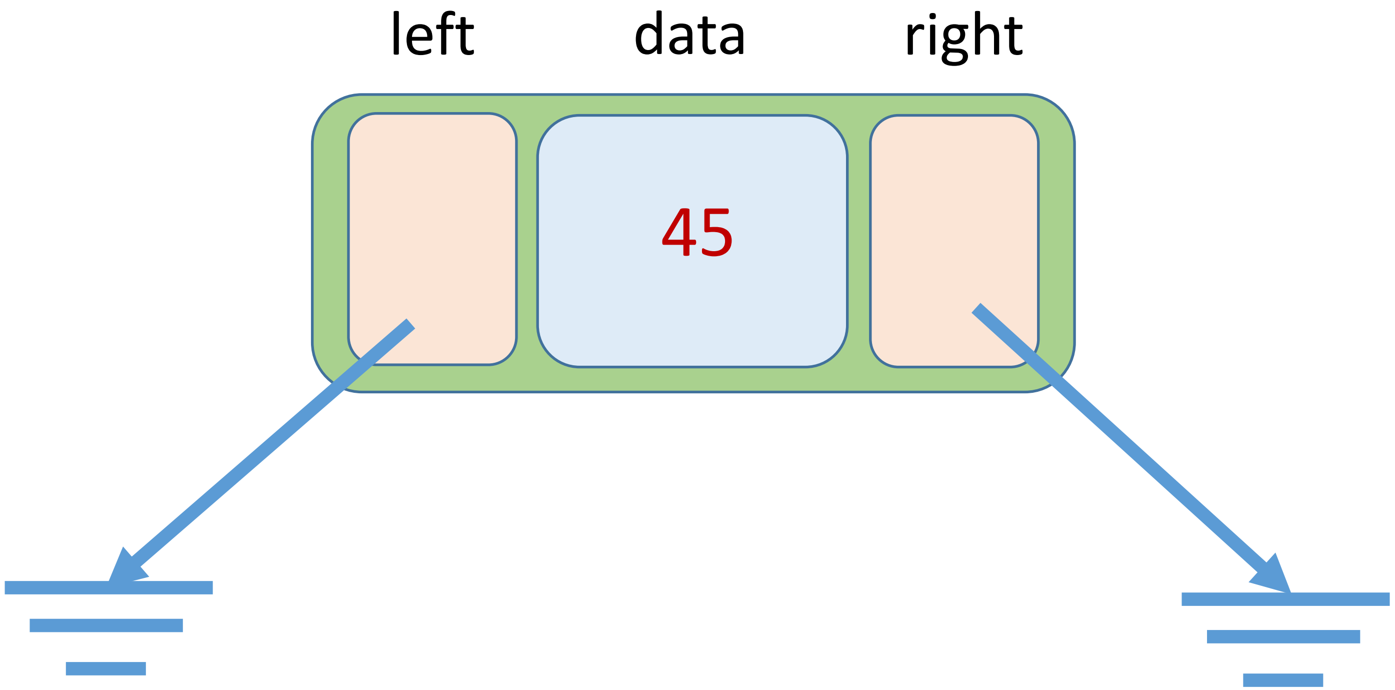
Give the Preorder, Inorder, and Postorder Traversals of This Binary Tree:



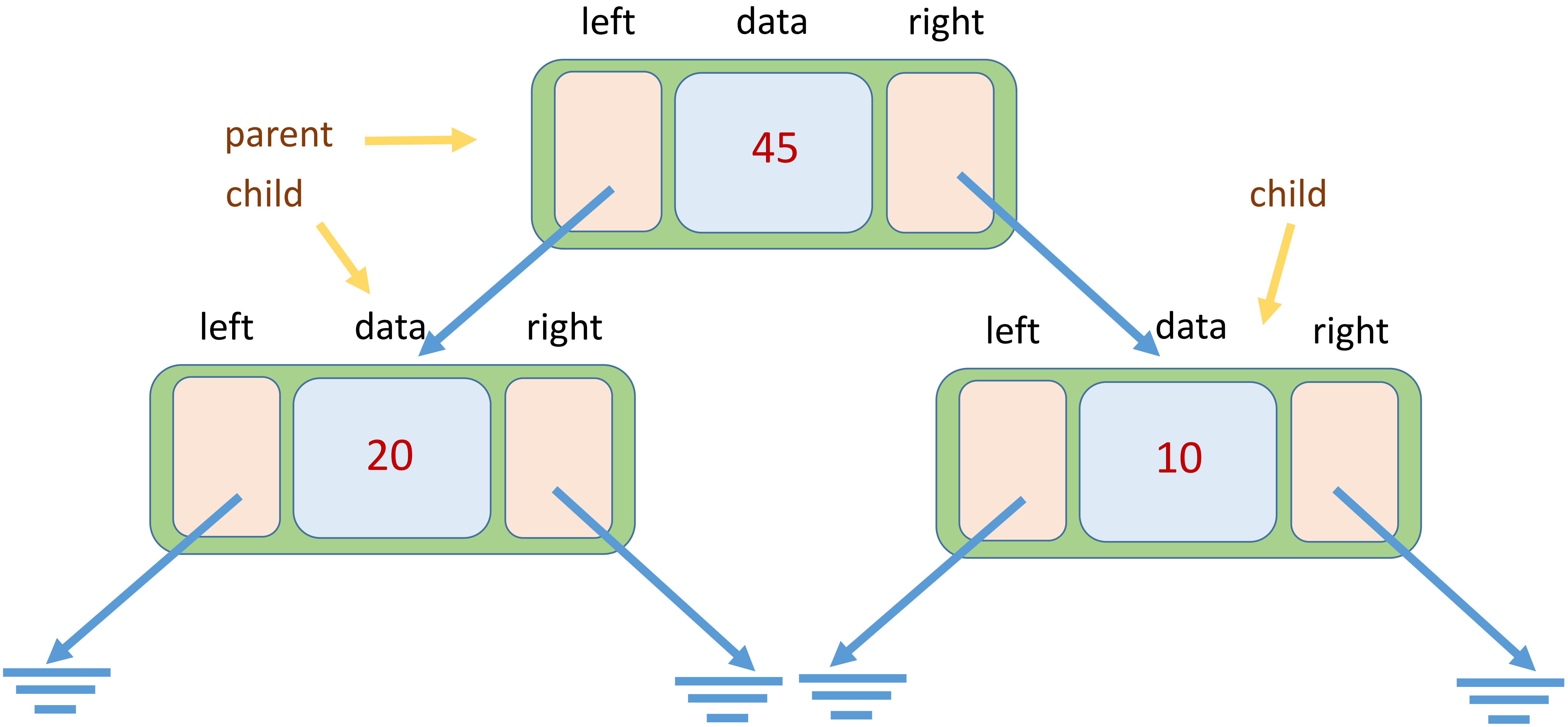
Implementation in C++

We will now discuss how to implement a binary tree and its related algorithms in C++.

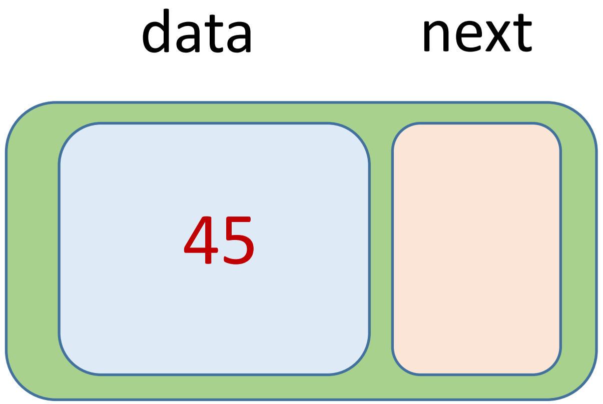
A Node in a Binary Tree



Implementation of Nodes in a Binary Tree

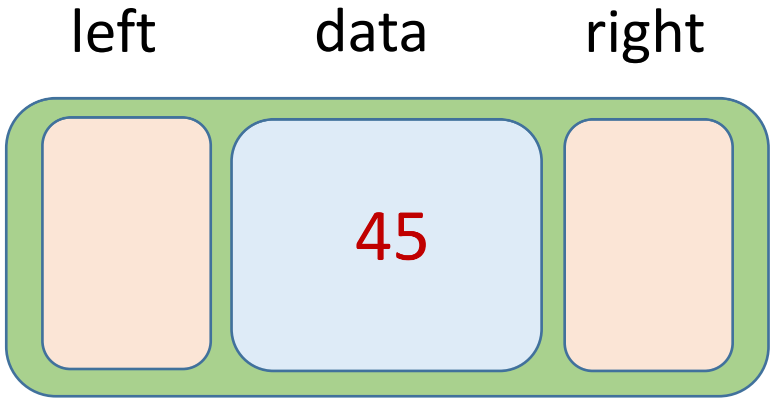


Declaring a Node in a Linked List



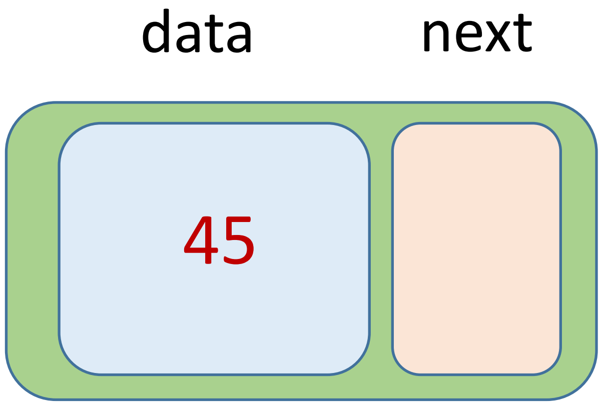
```
struct LLNode {
        int data;
        LLNode * next;
};
```

Declaring a Node in a Binary Tree



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

Declaring a Node in a Linked List



```
struct LLNode {
        int data;
        LLNode * next;
};
```

Exercise

Wrte the code for the *preOrder*, *inOrder*, and *postOrder* functions with the following prototypes:

```
void preOrder (BTNode * root);
void inOrder (BTNode * root);
void postOrder (BTNode * root);
```

The functions must all be recursive and should simply display the value stored in the node when it is “visited”.

Solution for Exercise (preOrder)

```
void preOrder (BTNode * root) {

     if (root == NULL)
       return;






}
```

Solution for Exercise (preOrder)

```
void preOrder (BTNode * root) {

     if (root == NULL)
       return;

     cout << root->data << endl;

     preOrder (root->left);
     preOrder (root->right);

}
```