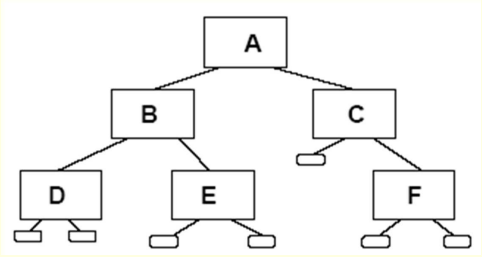
Lecture 11.2

Traversing Tree

Binary Tree representation



- Each node is either an empty tree or
- A binary tree with a left and a right node, each of which is a binary tree.

Tree Traversal

- Just like in linked lists we traverse the list by visiting every node in the list, we can find a way to visit every node in the tree.
- Unlike traversing the list, the best way to visit each item in a tree in an orderly fashion is not so obvious
- Where should you start?
  - At the root, maybe?
- Where should you proceed?

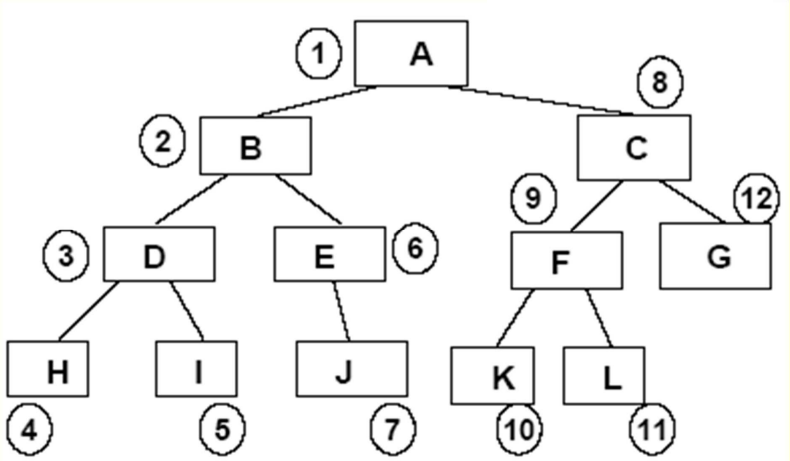
Traversal schemes

- These schemes take advantage of the recursive nature of the tree.
- The basic idea is:
  - visit the root directly
  - visit the children recursively
  - If we find an empty branch (i.e. tree pointer pointing to NULL), no action is required
- so this serves as the base case for the recursion.

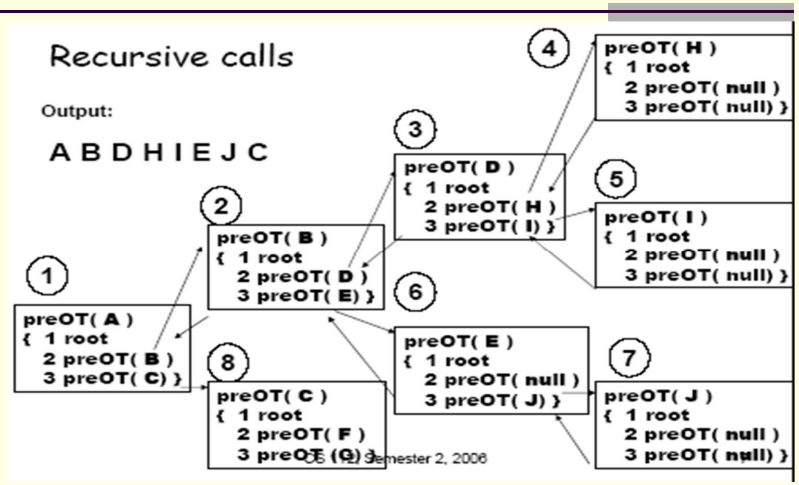
Preorder Traversal

- If the tree is not NULL
  1. visit the root
  2. preOrderTraverse (left child)
  3. preOrderTraverse (right child)

Preorder Traversal



Preorder Traversal



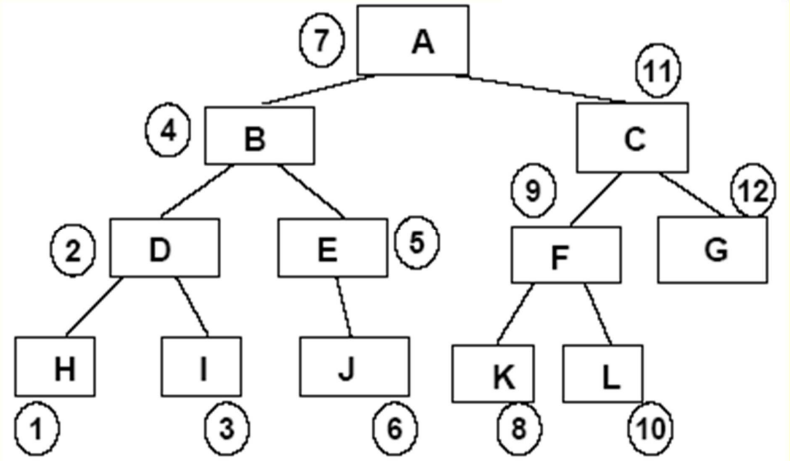
Preorder Traverse code

```
template <class dataType>
void preOrderTraverse (BinaryTreeNode <dataType>* bt)
{
    if ( ! (bt == NULL) )
    {
        //visit tree
        cout << bt -> getData ( ) <<"\t";
        //traverse left child
        preOrderTraverse ( bt -> left ( ) );
        //traverse right child
        preOrderTraverse ( bt -> right ( ) );
    }
}
```

Inorder Traversal

- If the tree is not NULL
  1. inOrderTraverse (left child)
  2. visit the root
  3. inOrderTraverse (right child)

Inorder Traversal



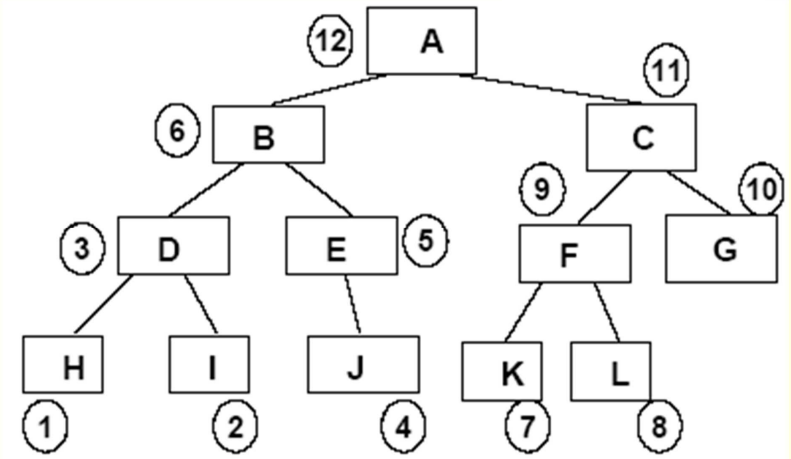
Code for InorderTraverse

```
template <class dataType>
void inOrderTraverse (BinaryTreeNode <dataType> * bt)
{
    if ( ! (bt == NULL) )
    {
        //traverse left child
        inOrderTraverse ( bt -> left ( ) );
        //visit tree
        cout << bt->getData ( ) <<"\t";
        //traverse right child
        inOrderTraverse ( bt -> right ( ) );
    }
}
```


Postorder Traversal

- If the tree is not NULL
  1. postOrderTraverse (left child)
  2. postOrderTraverse (right child)
  3. visit the root

Postorder Traversal



Postorder traversal code

```
template <class dataType>
void postOrderTraverse(BinaryTreeNode <dataType> * bt)
{
    if ( ! (bt == NULL) )
    {
            //traverse left child
            postOrderTraverse ( bt->left ( ) );
            //traverse right child
            postOrderTraverse ( bt -> right ( ) );
            //visit tree
            cout << bt -> getData ( ) <<"\t";
    }
}
```