



**Sanjivani Rural Education Society's**  
**Sanjivani College of Engineering, Kopargaon-423 603**  
*(An Autonomous Institute, Affiliated to Savitribai Phule Pune University, Pune)*  
*NACC 'A' Grade Accredited, ISO 9001:2015 Certified*

## **Department of Computer Engineering**

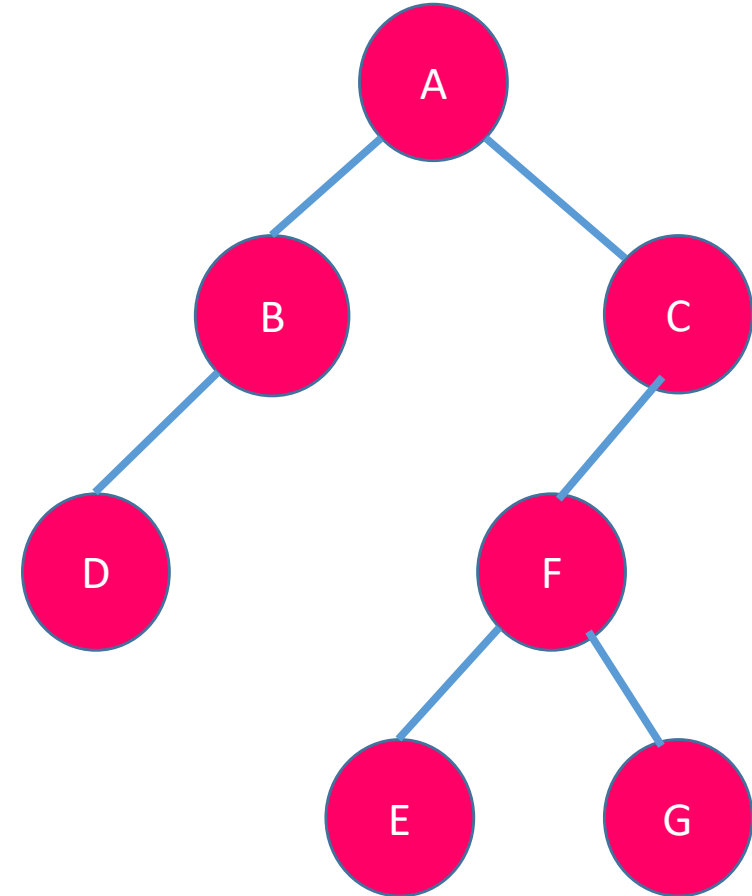
*(NBA Accredited)*

**Subject- Data Structures**  
**Unit-I Tree**  
**Binary Tree and Operations-1**

**Prof. Nirgude V.N.**  
**Assistant Professor**  
**E-mail : [nirgudevikascomp@sanjivani.org.in](mailto:nirgudevikascomp@sanjivani.org.in)**  
**Contact No: 9975240215**

# Binary Tree

- In binary tree, every node can have at most two branches i.e. there is no node with degree greater than two.
- Definition:
  - A binary tree is a finite set of nodes, which is either empty or consist of a T and two disjoint binary tree called as left sub tree and the right sub tree.

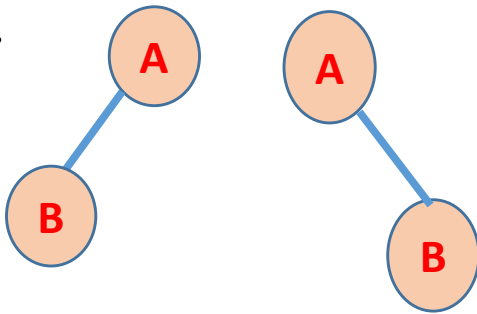


- Difference between tree and binary tree

1. For every node may have left sub tree and right sub tree whereas in tree sub tree doesn't matter. (Order does not matter in tree)

2. Binary tree can have zero nodes i.e. binary tree can be empty, which is not in case of tree

3.



In this example 1<sup>st</sup> binary tree has empty right sub tree while second binary tree has empty left tree. if we consider as tree then both are same only representation is different.

Aspect	Tree	Binary Tree
Definition	A hierarchical structure where each node can have any number of children.	A hierarchical structure where each node can have at most two children (left and right).
Root Node	Always present in a non-empty tree.	Can be absent as a binary tree can be empty.
Number of Children	A node can have any number of children.	A node can have at most two children (0, 1, or 2).
Order of Children	No inherent order among the children of a node.	The order matters (left and right children are distinct).
Structure Flexibility	More flexible with no strict rules on the number of children per node.	More structured with a maximum of two children per node.

Traversal	Uses general traversal methods like DFS and BFS.	Includes specific traversals like In-Order, Pre-Order, and Post-Order.
Types	General Tree, N-ary Tree, etc.	Full Binary Tree, Complete Binary Tree, Balanced Binary Tree, etc.
Applications	Used in file systems, organizational charts, XML/HTML parsing, etc.	Used in searching (Binary Search Tree), heaps, and decision-making algorithms.
Empty Structure	Cannot be empty; must have at least a root node.	Can be empty with zero nodes.
Storage Complexity	Depends on the number of nodes and the structure.	Often optimized due to the fixed two-children constraint.

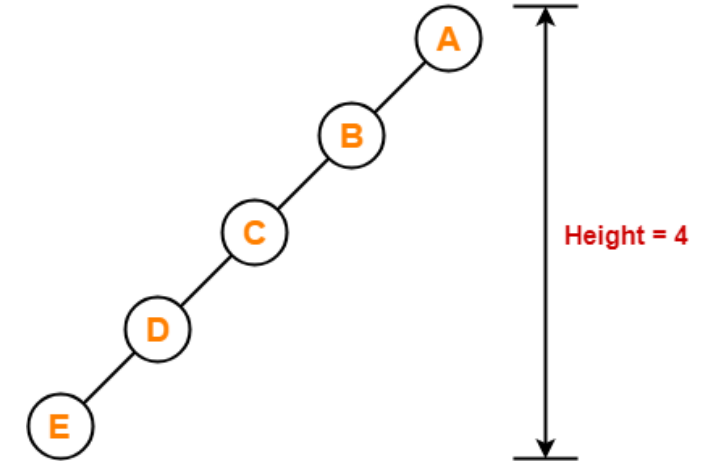


# Properties of Binary Tree

1. Minimum number of nodes in a binary tree of height  $H = H + 1$

Example-

To construct a binary tree of height = 4, we need at least  $4 + 1 = 5$  nodes.



2. Maximum number of nodes in a binary tree of height  $H = 2^{H+1} - 1$

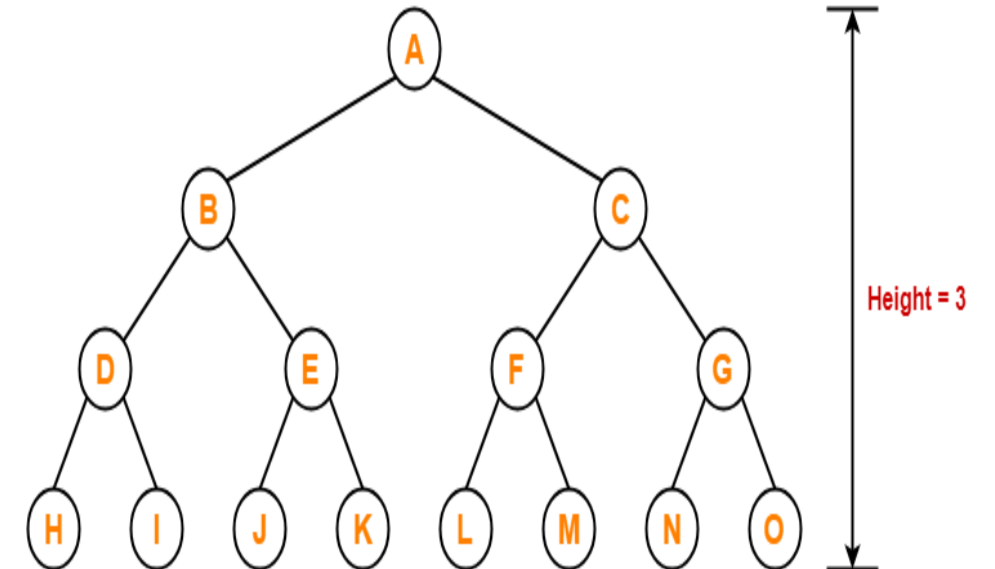
Example-

Maximum number of nodes in a binary tree of height 3

$$= 2^{3+1} - 1$$

$$= 16 - 1$$

$$= 15 \text{ nodes}$$

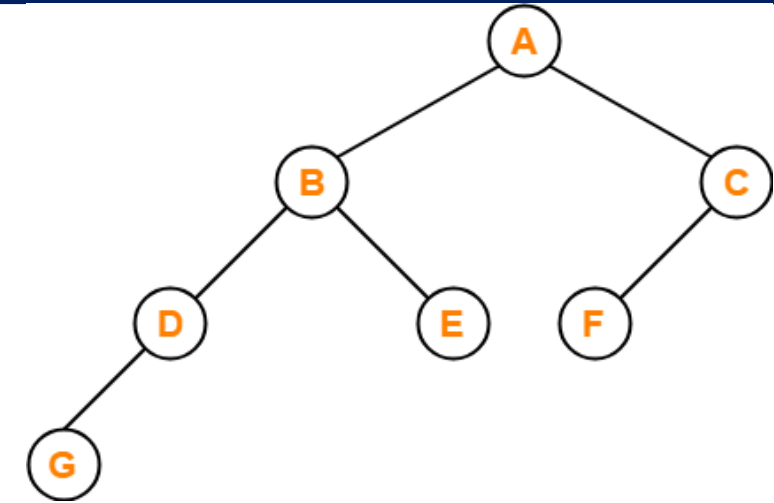


### 3. Total Number of leaf nodes in a Binary Tree

= Total Number of nodes with 2 children + 1

Here, Number of leaf nodes = 3, Number of nodes with 2 children = 2

•Clearly, number of leaf nodes is one greater than number of nodes with 2 children.



### 4. Maximum number of nodes at any level 'L' in a binary tree= $2^L$

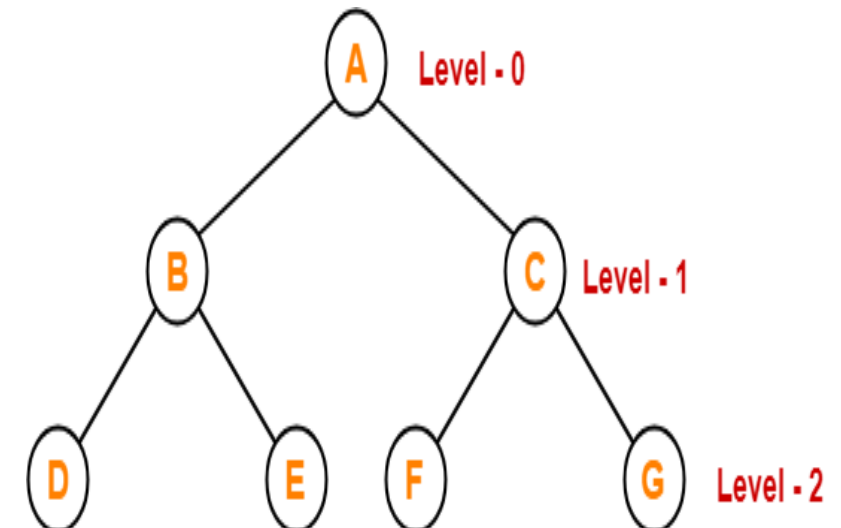
#### Example-

Maximum number of nodes at level-2 in a binary tree

$$= 2^2$$

$$= 4$$

Thus, in a binary tree, maximum number of nodes that can be present at level-2 = 4.



1. The height of a binary tree that contains  $n$ ,  $n \geq 0$  element is atmost  $n$  and atleast  $\lceil \log_2(n+1) \rceil$

example:  $\log_2(n+1)$  if  $n=15$

$$= \log_2 (15+1) = \log(16) / \log(2)$$

$$= 4$$

$$(n \leq 2^h - 1)$$





# Exercise

1. A binary tree T has n leaf nodes. The number of nodes of degree-2 in T is \_\_\_\_\_?

1.  $\log_2 n$    2.  $n-1$    3.  $n$    4.  $2^n$

2. In a binary tree, the number of internal nodes of degree-1 is 5 and the number of internal nodes of degree-2 is 10. The number of leaf nodes in the binary tree is \_\_\_\_\_?

1. 10   2. 11   3. 12   4. 15

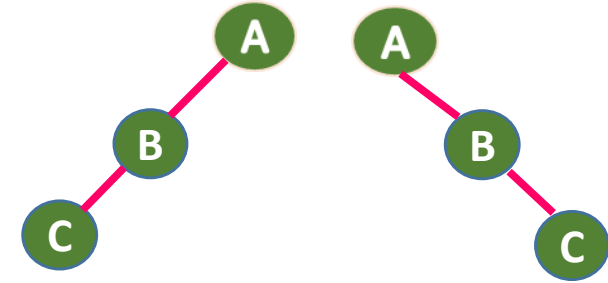
3. A binary tree T has 20 leaves. The number of nodes in T having 2 children is \_\_\_\_\_?

1. 20   2. 10   3. 19   4. 15

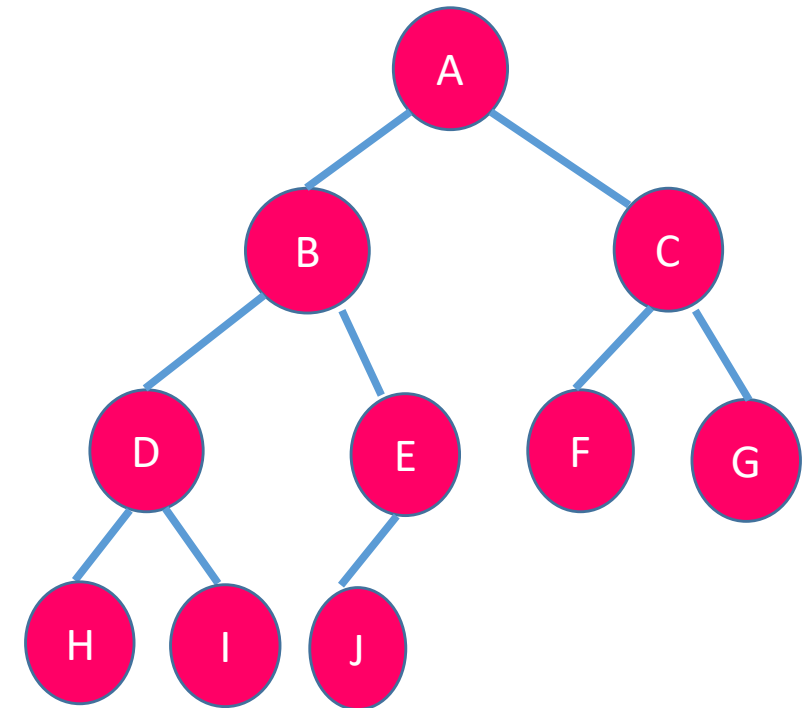


# Type of Binary Tree

1. **Skewed Binary Tree:** a binary tree in which every node is having either only left sub tree or right sub tree

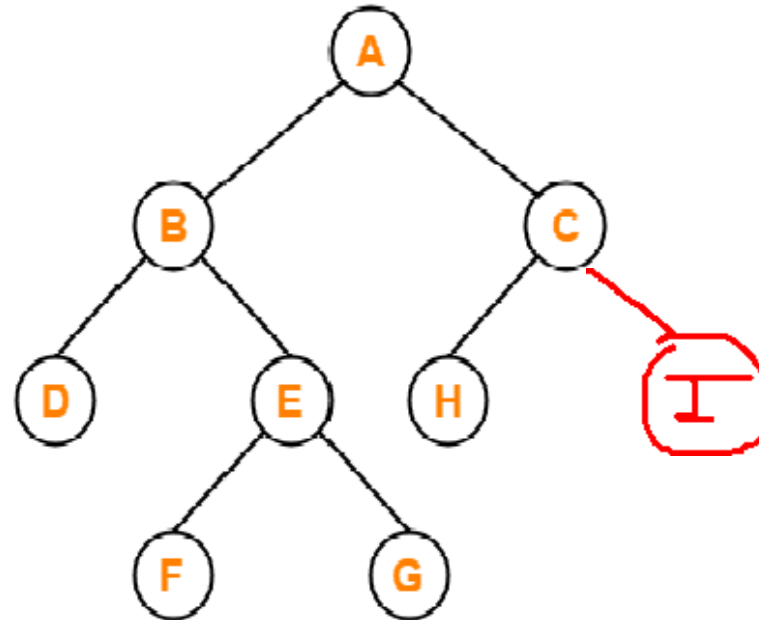
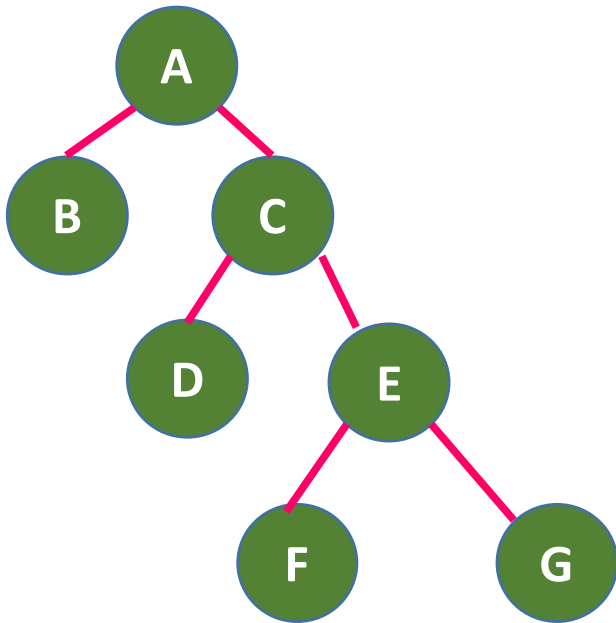


2. **Almost Complete Binary Tree:** In a complete binary tree, each non-leaf node compulsory has sub tree. Also, in the last or the lowest level of this binary tree, every node should possibly reside on the left side. Here is the structure of a complete binary tree:



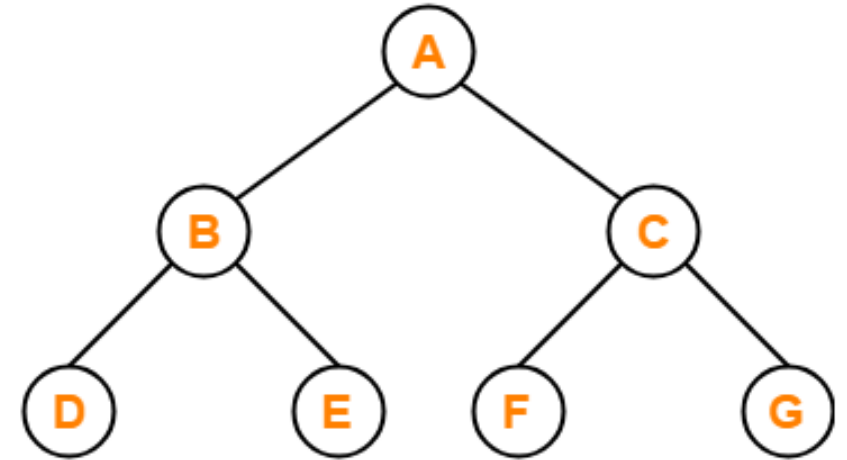
**Strictly Binary Tree:** if every non-terminal node in a binary tree consist of non-empty left sub tree and right sub tree then such tree is called as strictly binary tree.

- In other words internal node will have either two children or no child at all.




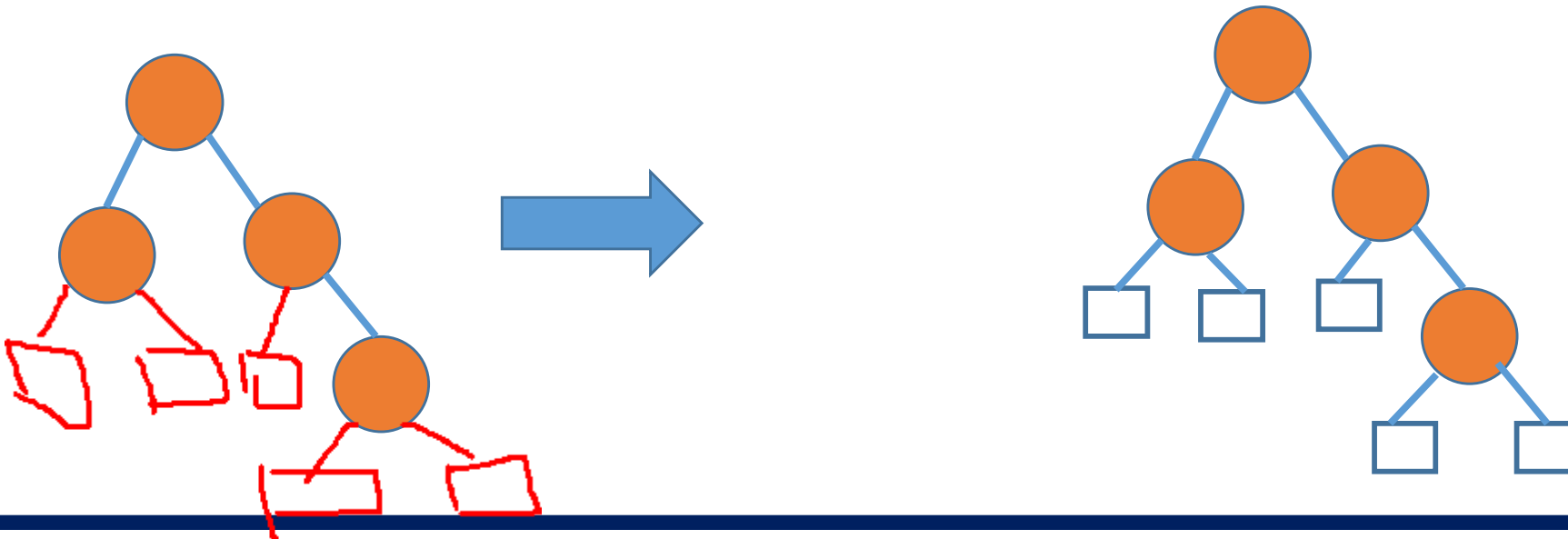
## Complete Binary Tree/Perfect Binary Tree:

- A **complete binary tree** is a binary tree that satisfies the following 2 properties-
- Every internal node has exactly 2 children.
- All the leaf nodes are at the same level.
- Complete binary tree is also called as **Perfect binary tree**.



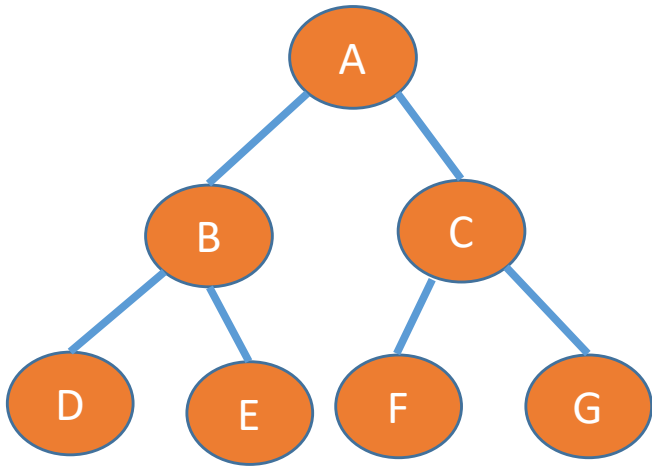
- **Extended Binary Tree:**

- Each empty sub tree is replaced by a failure node. A failure node is represented by 
- Any binary tree can be converted into a extended binary tree by replacing each empty sub tree by a failure nodes

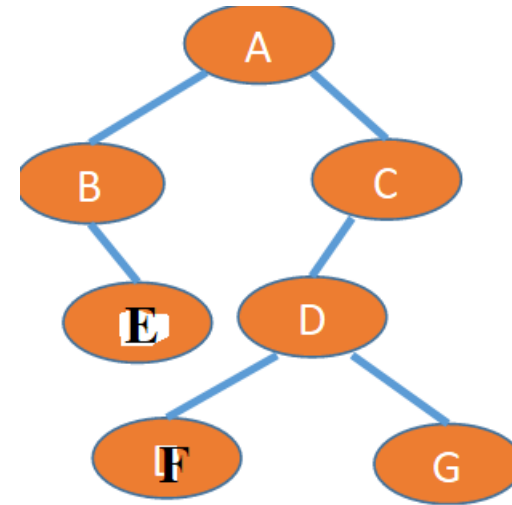


# Representation of Binary Tree

- To represent the binary tree in one dimensional array, we need to number the nodes sequentially level by level(left to right).
- Every empty nodes are also numbered.



7	A	B	C	D	E	F	G
---	---	---	---	---	---	---	---



13	A	B	C	-	G	D	-	-	-	-	-	E	F
----	---	---	---	---	---	---	---	---	---	---	---	---	---



- For complete Binary tree there is no issue, but for skew tree there is a lot of wastage of space. e.g  $k$  depth of skew requires  $2^k - 1$  space out of only  $k$  get occupied in array.
- Therefore another way is needed to represent the Binary tree. That is nothing but linked representation which is an efficient way than array.



# Binary Tree Node

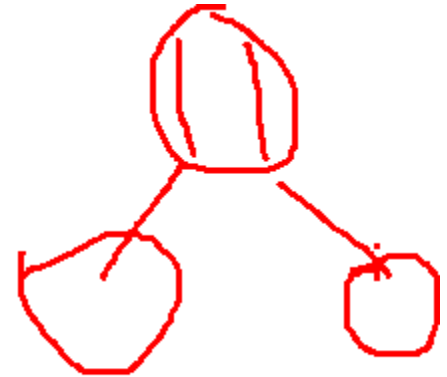
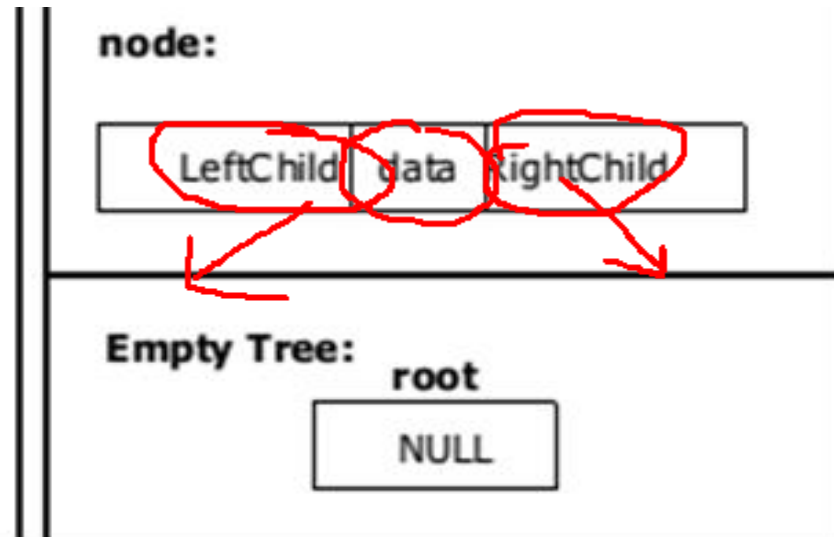
- The code to write a tree node would be similar to what is given below. It has a data part and references to its left and right child nodes.

In a tree, all nodes share common construct.

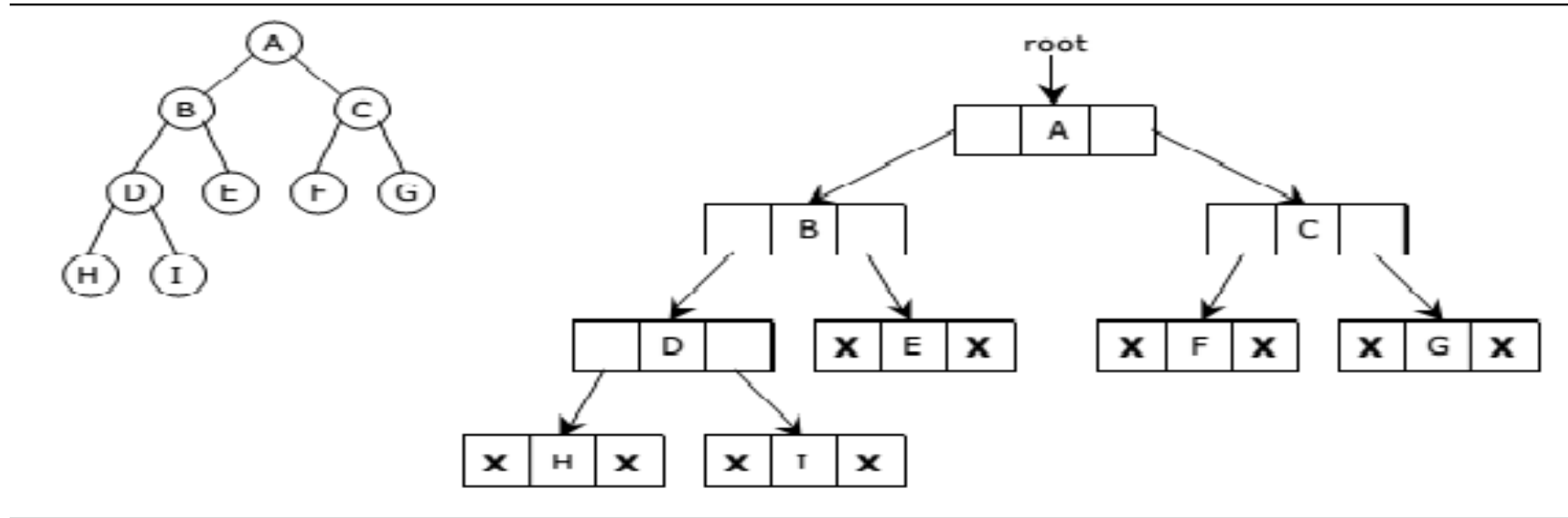
```
struct binarytree
{
    struct binarytree *LeftChild;
    int data;
    struct binarytree *RightChild;
};

typedef struct binarytree node;

node *root = NULL;
```



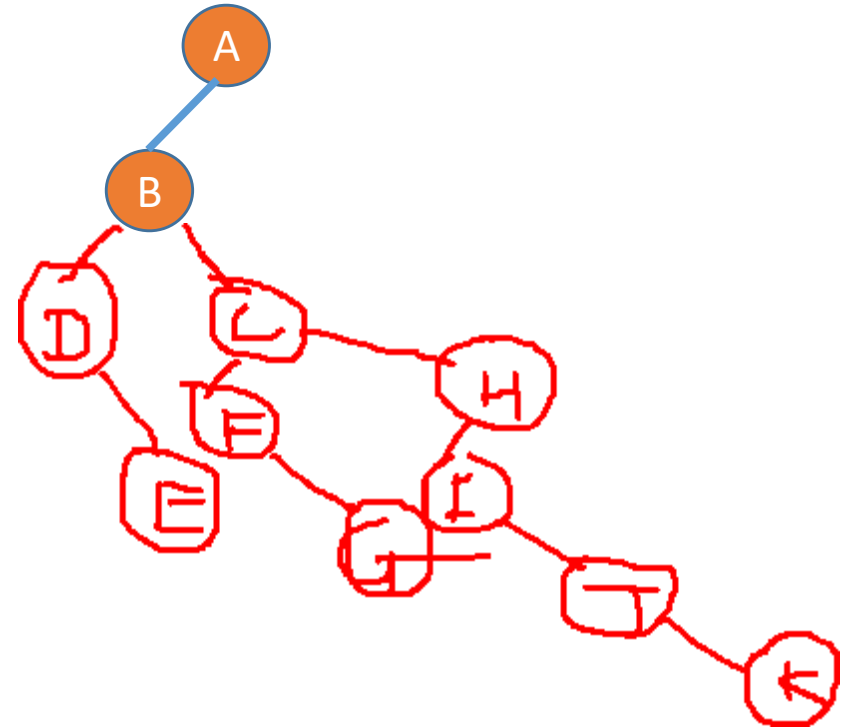
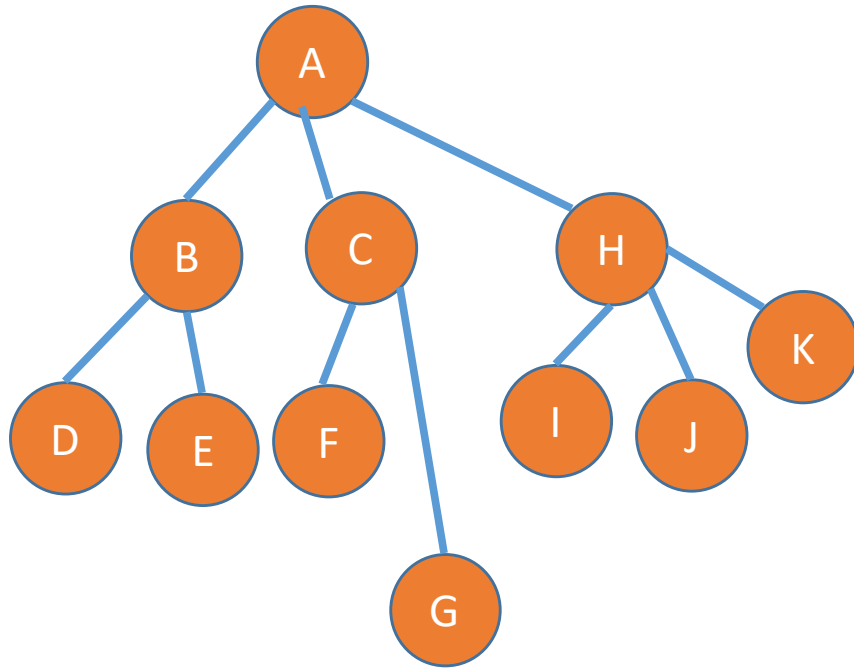




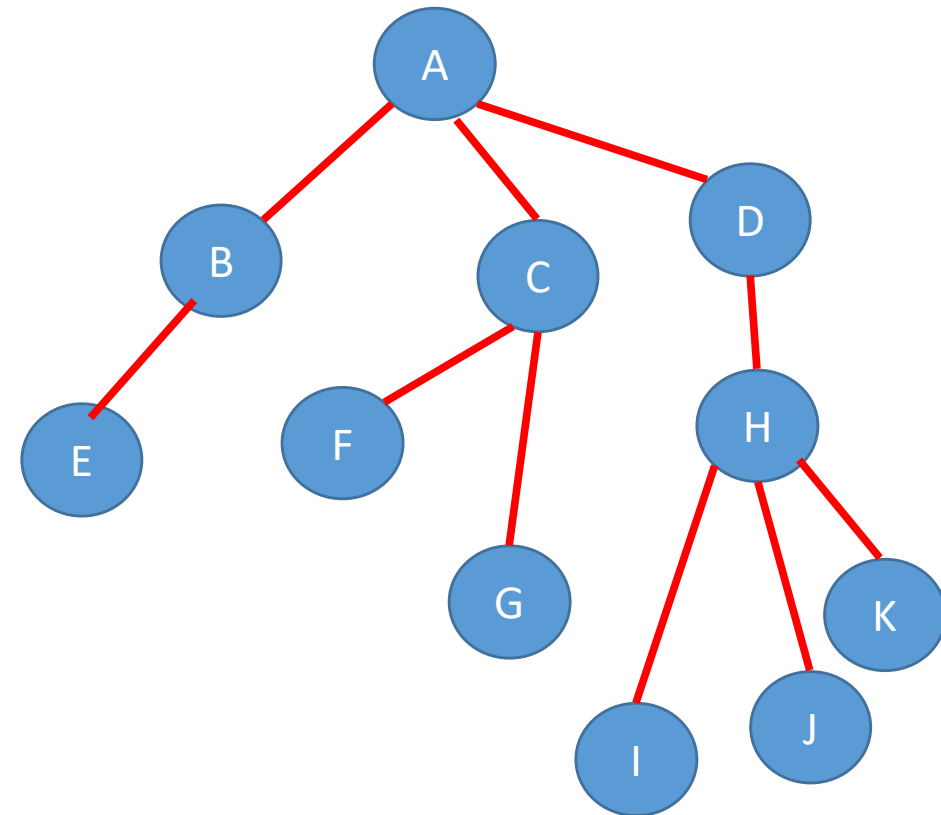
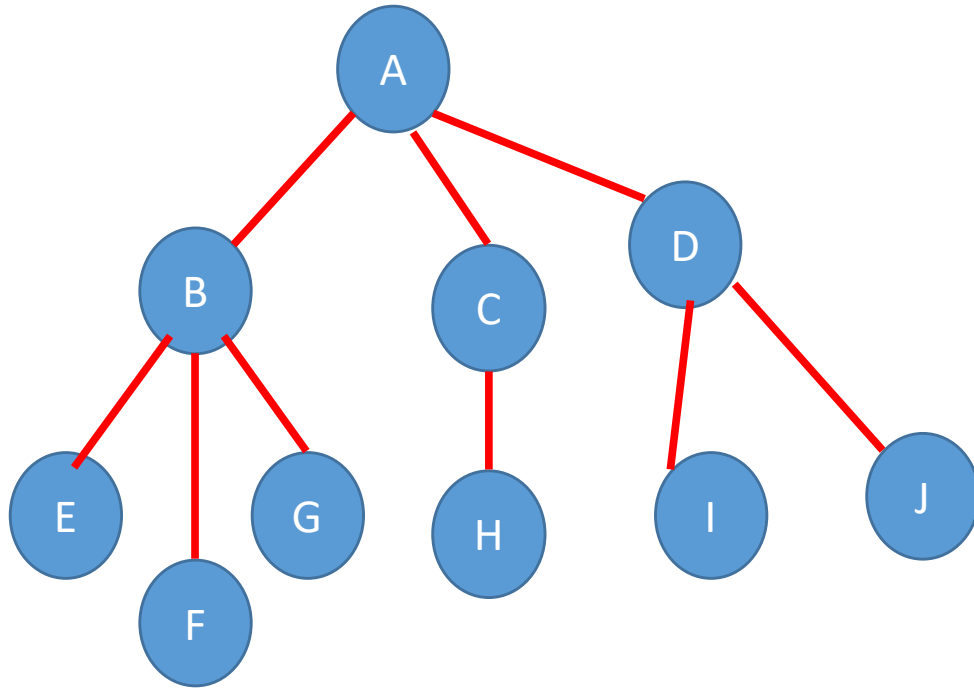
Linked representation of Binary Tree

# Conversion of Tree into Binary Tree

- Children of parents added using leftmost child's right sibling relation.



# Examples for Exercise





# Binary tree Basic Operations

---

The basic operations that can be performed on a binary tree data structure, are the following –

**Insert** – Inserts an element in a tree/create a tree.

**Traversals** – Searches an element in a tree.

**Preorder Traversal** – Traverses a tree in a pre-order manner.

**Inorder Traversal** – Traverses a tree in an in-order manner.

**Postorder Traversal** – Traverses a tree in a post-order manner.

**Search-** Search an element in tree using traversals.

**Delete-** delete an element from binary tree



# Create/Insert OPERATION

- The very first insertion creates the tree. Afterwards, whenever an element is to be inserted, first locate its proper location. Start searching from the T node, then search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.
- **Algorithm:**
  1. Enter key to be inserted. Create node as tempNode for it.
  2. Check T is NULL, if yes then make tempNode as T of tree
  3. If T is not NULL then
  4. Take temporary variable **\*ptr** and set **\*ptr=T**
  5. Do
    - i. Ask user in which direction user wants to insert data(Left or right)
    - ii. if direction is Left the check following condition



iii. if(ptr->left==NULL) //insert node at left of tree

```
{  
    ptr->left=tempNode;  
    break; }
```

```
else{  
    ptr=ptr->left;  
}
```

else //if user gives right direction

```
{  
    if(ptr->right==NULL)  
    {  
        ptr->right=tempNode;  
        break;  
    }
```

else

```
{  
    ptr=ptr->right;  
}  
}
```

iv. Do step 5 while(ptr!=NULL)

5. Repeat steps 1 to 5 until no more data to insert.

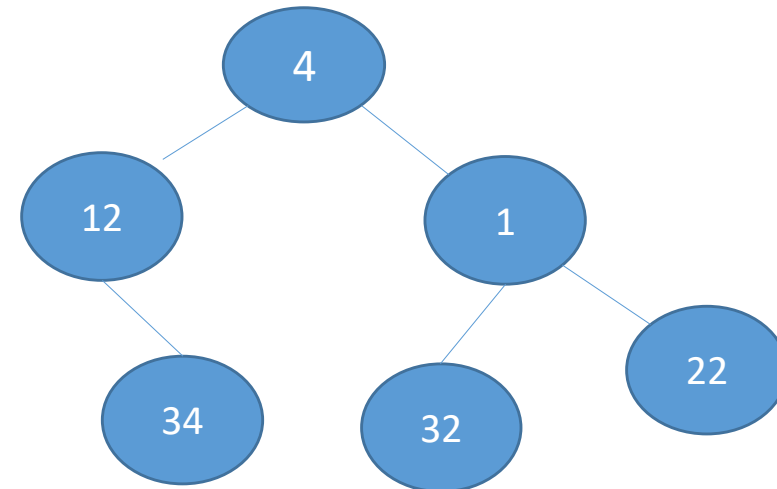
6. Stop



Btree.cpp

Enter the element=4

- Do u want to enter more elements (y/n)y
- Enter the element=12
- in which direction(l/r)l
- Do u want to enter more elements (y/n)y
- Enter the element=34
- in which direction(l/r)l
- in which direction(l/r)?r
- Do u want to enter more elements (y/n)y
- Enter the element=1
- in which direction(l/r)r
- Do u want to enter more elements (y/n)y
- Enter the element=22
- in which direction(l/r)r
- in which direction(l/r)?r
- Do u want to enter more elements (y/n)y
- Enter the element=32
- in which direction(l/r)r
- in which direction(l/r)?l





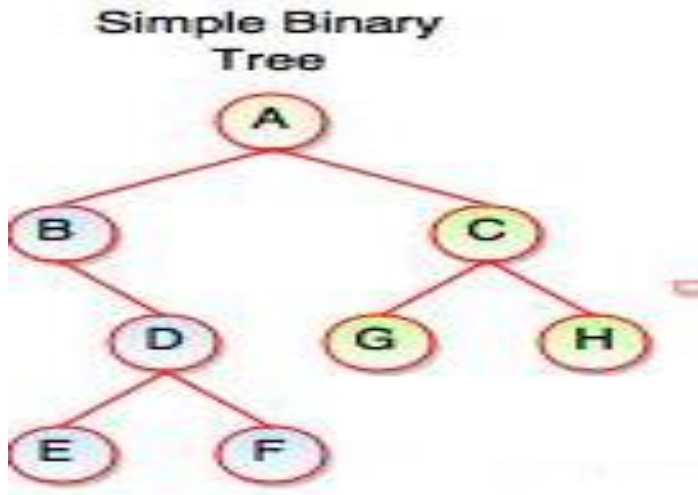
# Binary Tree Traversal

---

- Traversal is a process to visit all the nodes of a tree and print their values too. Because, all nodes are connected via edges (links) we always start from the T (head) node.
- This process could be defined recursively.
- We cannot access a node randomly from a tree. There are three ways which we use to traverse a tree –
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal



# 1. In - Order Traversal ( LeftChild - T - RightChild )



## Algorithm for inorder traversal

**Step 1 :** Start from the Left Subtree of T .

**Step 2 :** Then, visit the T.

**Step 3 :** Then, go to the Right Subtree.

**Step 1 :** Inorder on (B) + A+(Inorder on C)

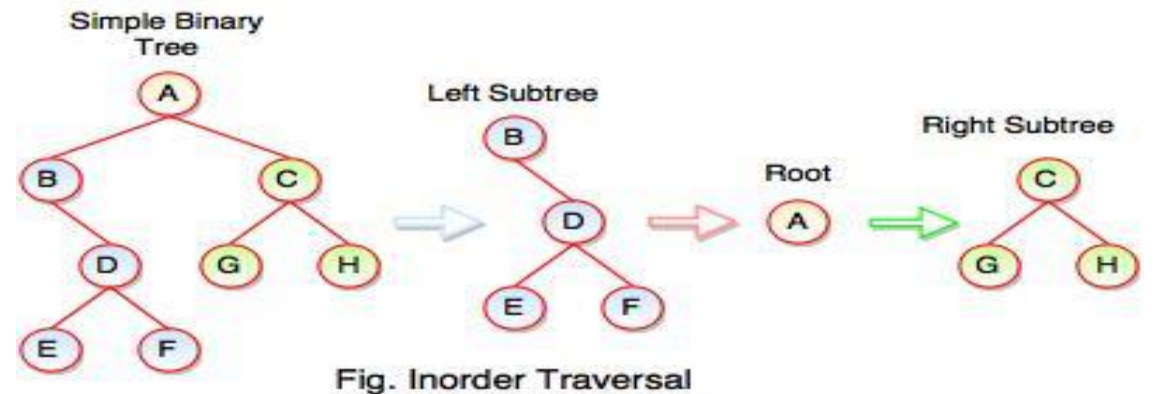
**Step 2 :** [B+ inorder on(D) ]+A+ (Inorder on C)

**Step 3 :** B + inorder on(E) + D + Inorder on( F )+ A +Inorder on (C)

**Step 4 :** B + E + D + F + A +Inorder on(G)+C+ Inorder on(H)

**Step 4 :** B + E + D + F + A +G+C+ H

**Inorder Traversal : B E D F A G C H**





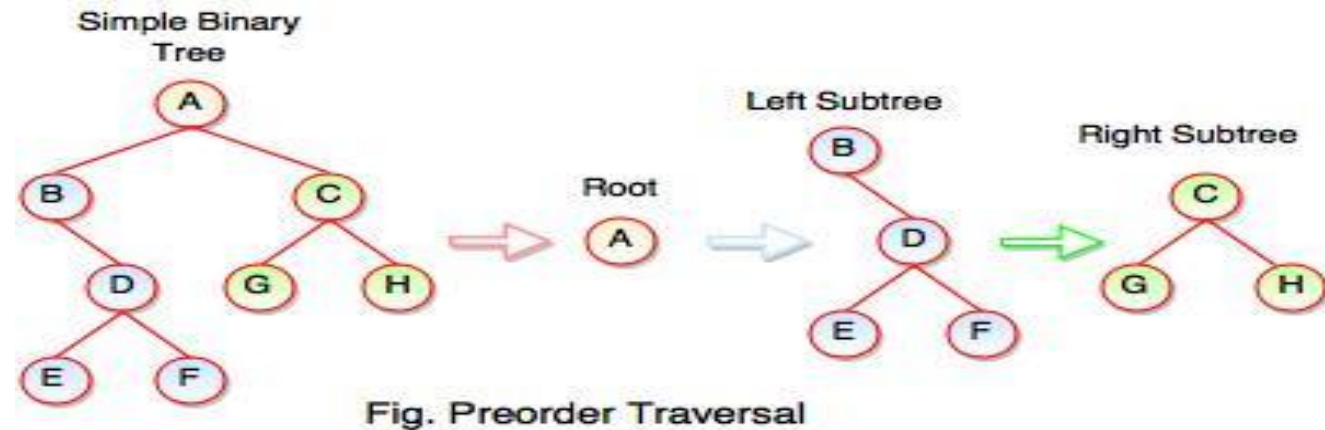
## 2. Pre - Order Traversal ( T - LeftChild - RightChild )

### Algorithm for preorder traversal

**Step 1 :** Start from the T and visit the T.

**Step 2 :** Then, go to the Left Subtree.

**Step 3 :** Then, go to the Right Subtree.



**Step 1 :** A + Preorder on (B) + Preorder on(C)

**Step 2 :** A + [B + Preorder on(D)] +Preorder on (C )

**Step 3 :** A + [B + [D + Preorder on(E )+Preorder on( F)]] + Preorder on (C )

**Step 4:** A+ B+ D+ E+ F+ [C+Preorder on(G)+Preorder on(H)]

**Step 5:** A + B + D+ E+ F + C+ G + H

**Preorder Traversal : A B D E F C G H**



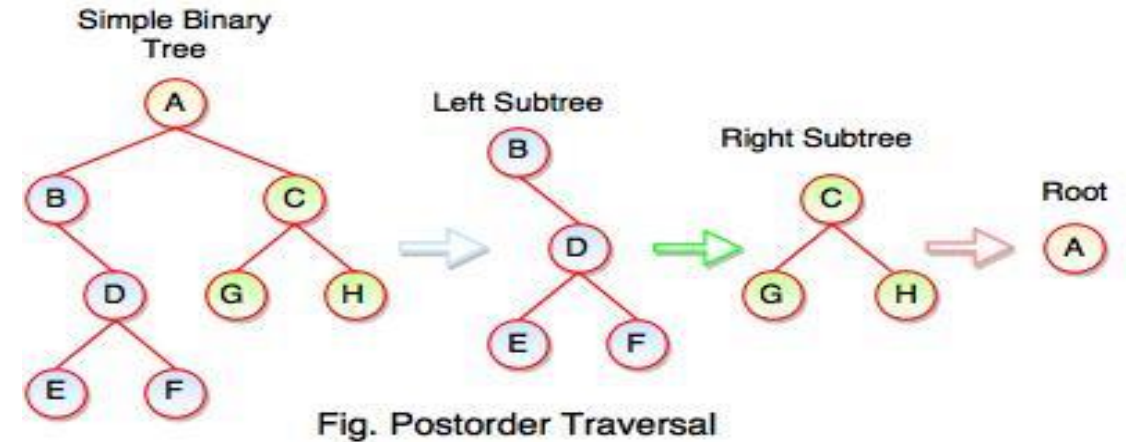
### 3. Post - Order Traversal ( Left-Child – Right-Child - T )

- Algorithm for post-order traversal

**Step 1 :** Start from the Left Subtree (Last Leaf) and visit it.

**Step 2 :** Then, go to the Right Subtree.

**Step 3 :** Then, go to the T.



**Step 1 :** (Postorder on (B) + Postorder on (C) + ( A)

**Step 2 :** [Postorder on(D)+ B]+ Postorder on (C) + ( A)

**Step 2 :** [[Postorder on(E)+Postorder on(F)+D]+ B]+ Postorder on (C) + ( A)

**Step 3 :** E + F + D + B + [[Postorder on(G)+ Postorder on(H)]+C]+A

**Step 3 :** E + F + D + B + G + H + C + A

**Post-order Traversal :** E F D B G H C A



```
void pre_order_traversal(struct node* T)
```

```
{
    if(T != NULL)
    {
        cout<<T->data;
        pre_order_traversal(T->leftChild);
        pre_order_traversal(T->rightChild);
    }
}
```

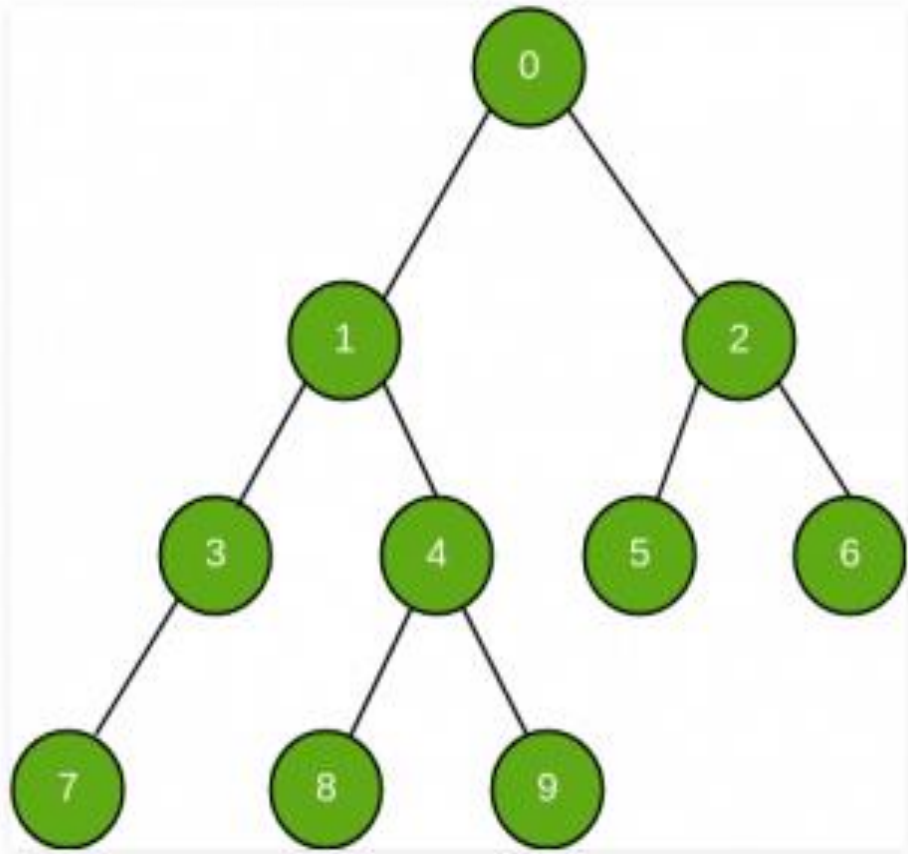
```
void inorder_traversal(struct node* T)
```

```
{
    if(T != NULL)
    {
        inorder_traversal(T->leftChild);
        cout<<T->data;
        inorder_traversal(T->rightChild);
    }
}
```

```
void post_order_traversal(struct node* T)
```

```
{
    if(T != NULL)
    {
        post_order_traversal(T->leftChild);
        post_order_traversal(T->rightChild);
        cout<<T->data;
    }
}
```

# Search OPERATION



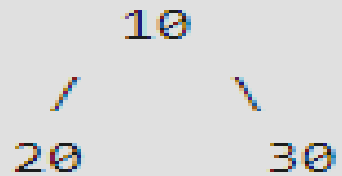
- The idea is, to use any of the tree traversals to search the node in the tree
- while traversing tree, check if the current node matches with the given node.
- Print YES if any node matches with the given node and stop traversing further
- and if the tree is completely traversed and none of the node matches with the given node then print NO.



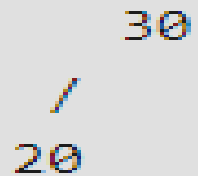
# DELETE OPERATION

- Deletion in a Binary Tree
- Given a binary tree, delete a node from it by making sure that tree shrinks from the bottom (i.e. the deleted node is replaced by bottom most and rightmost node). Examples :

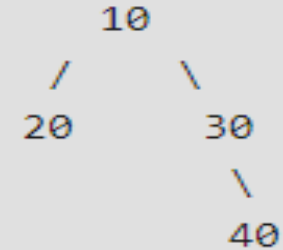
Delete 10 in below tree



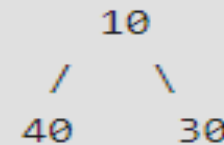
Output :



Delete 20 in below tree



Output :





- **Algorithm**

1. Starting at T, find the deepest and rightmost node in a binary tree and node which we want to delete.
2. Replace the deepest rightmost node's data with node to be deleted.
3. Then delete the deepest rightmost node.

