Trees

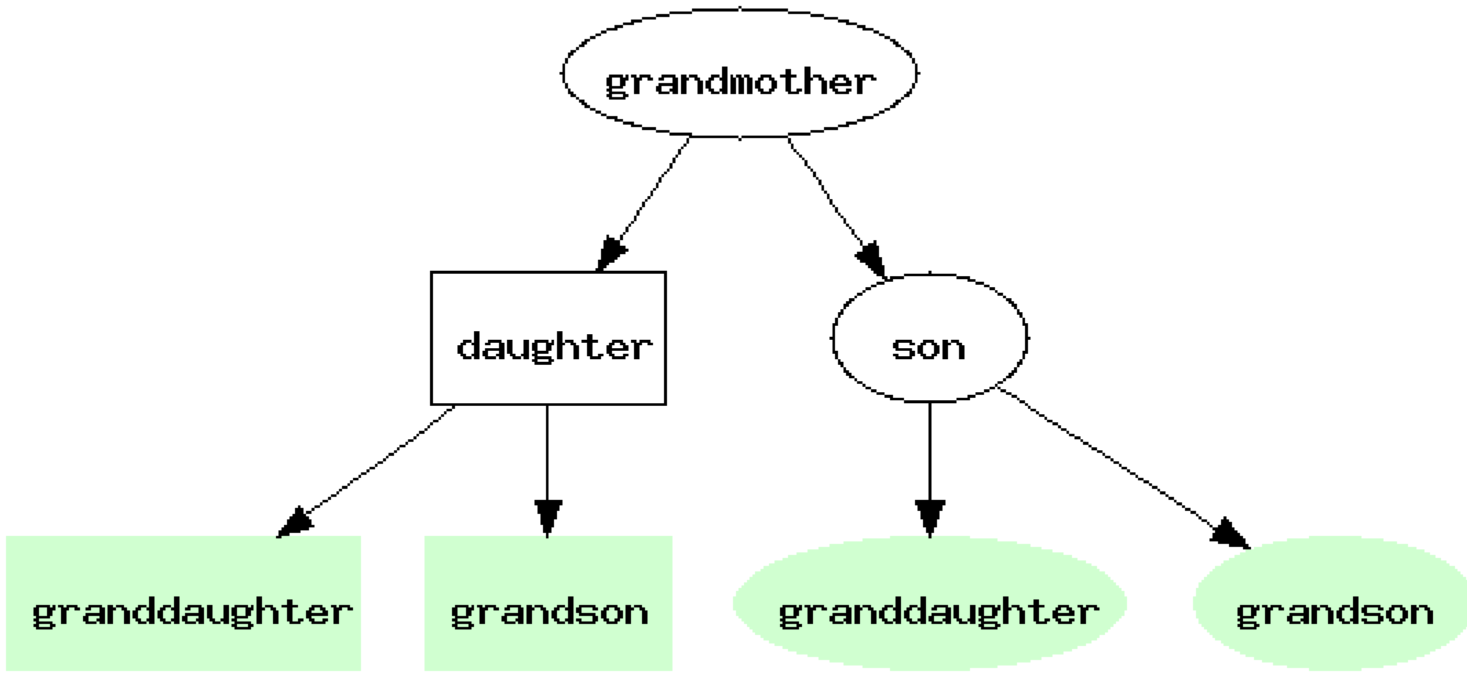
Trees

- Natural structures for representing certain kinds of hierarchical data.(How our files get saved under hierarchical directories)
- Allows us to associate a parent-child relationship between various pieces of data and allows arrange our records, data and files in a hierarchical fashion.
- Have many uses in computing. For example a *parse-tree* can represent the structure of an expression.
- Binary Search Trees help to order the elements in such a way that the searching takes less time as compared to other data structures.(speed advantage over other D.S)

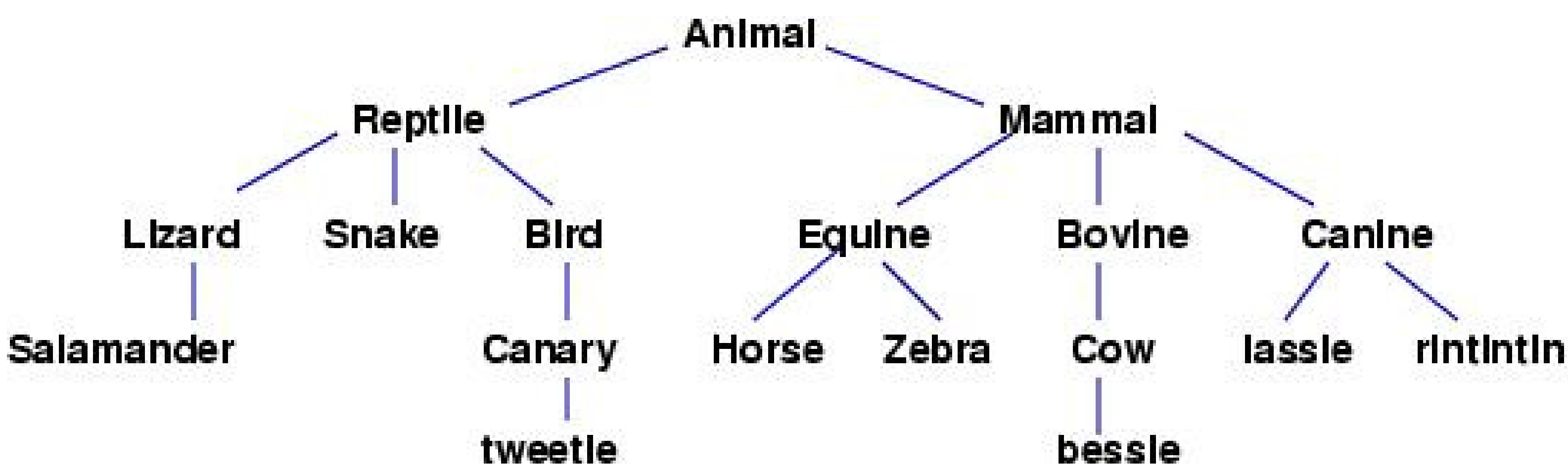
Trees

- Linked list is a linear D.S and for some problems it is not possible to maintain this linear ordering.
- Using non linear D.S such as trees and graphs more complex relations can be expressed.

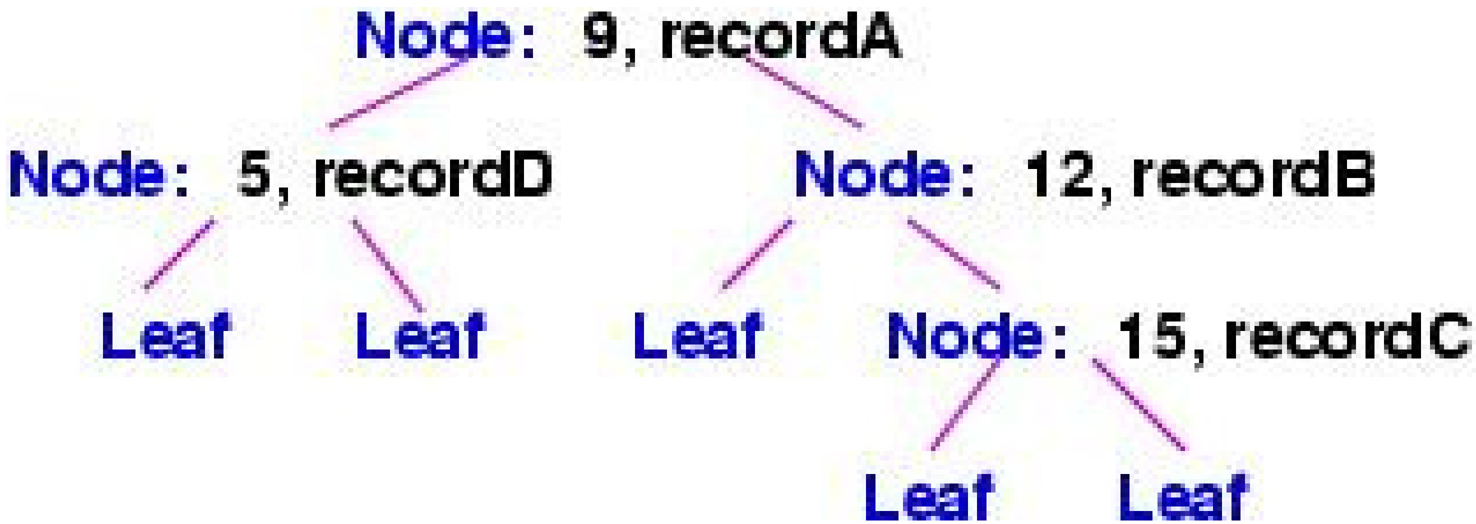
A Family Tree



tree of species, from zoology



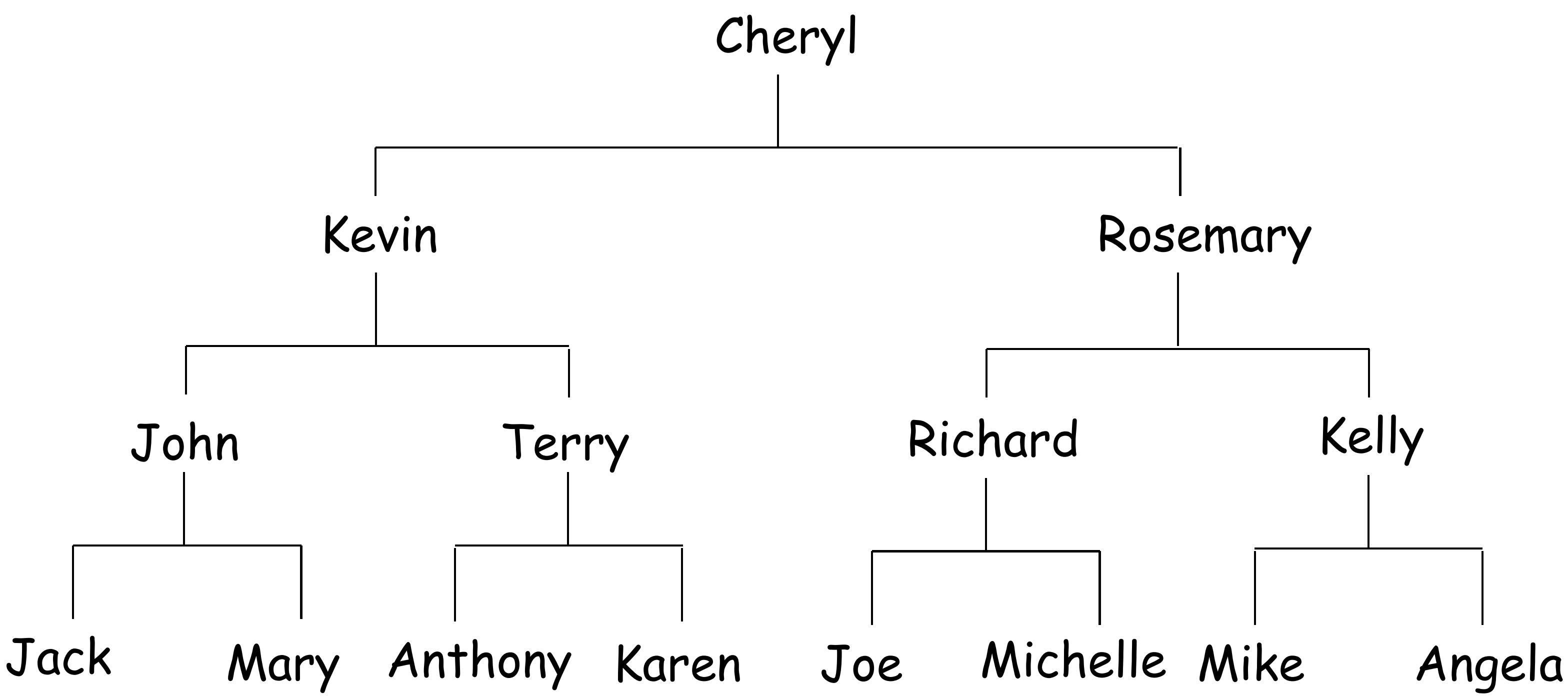
Ordered Tree or Search Tree



Parse Tree

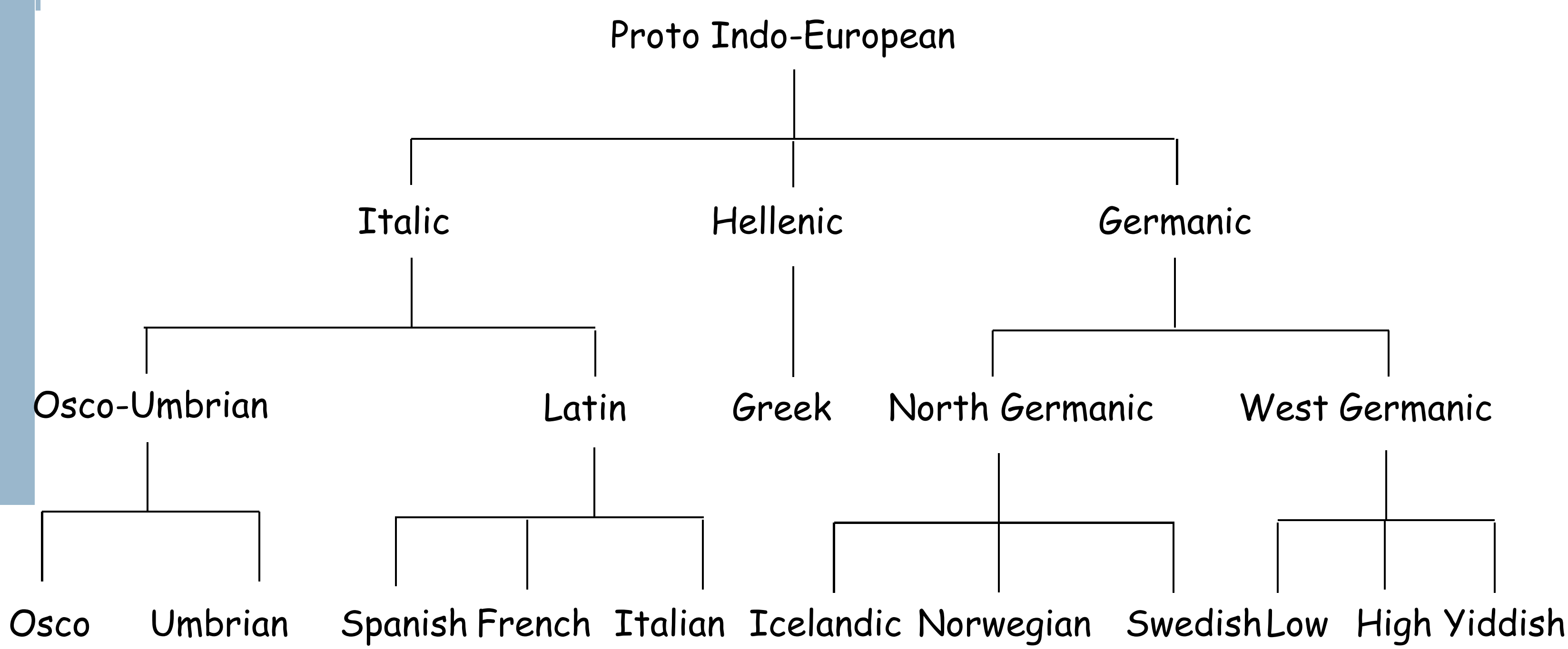
```
int x;
x = 3 + y;


                  STATEMENT SEQUENCE
                 /                  \
  DECLARATION                        ASSIGNMENT
   /         \                       /       \
TYPE  VARIABLE              VARIABLE     EXPRESSION
  |          |                  |          /     |   \
int          x                  x       NUMERAL  + VARIABLE
                                           |          |
                                           3          y
```

Pedigree Genealogical Chart



Binary Tree

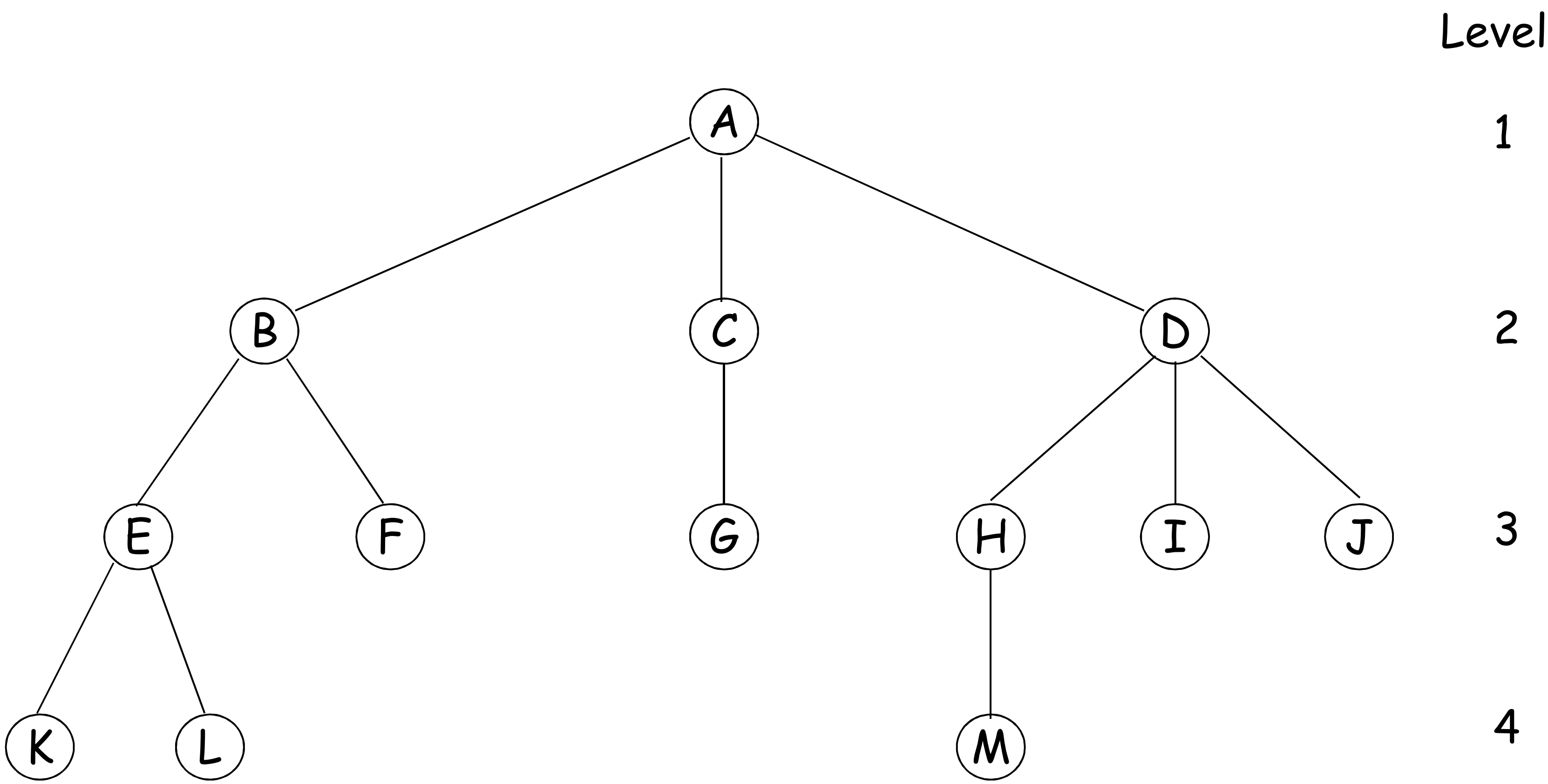
Lineal Genealogical Chart



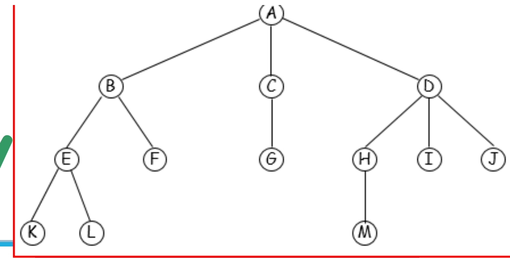
Trees

- Definition: A tree is a finite set of one or more nodes such that:
  - There is a specially designated node called the root.
  - The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, \ldots, T_n$, where each of these sets is a tree. We call $T_1, \ldots, T_n$ the subtrees of the root.

A Sample Tree



Tree Terminology

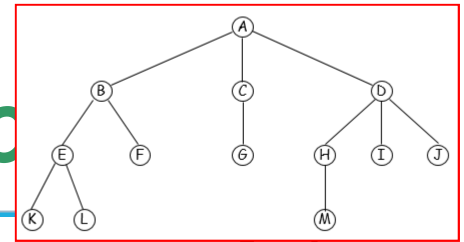


- Normally we draw a tree with the root at the top.
- A node stands for the item of information plus the branches to other nodes.
- The degree of a node is the number of subtrees of the node. [Degree of A=3, C=1, F=0]
- A node with degree zero is a leaf or terminal node.

  [K L F G M I J]
- A node that has subtrees is the parent of the roots of the subtrees, and the roots of the subtrees are the children of the node.

  [Children of B = E and F, parent of B is A]
- Children of the same parents are called siblings. [E and F]

Tree Terminology (Co



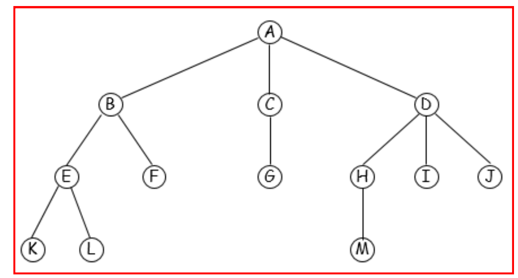
- The degree of a tree is the maximum degree of the nodes in the tree. [Degree of above tree = 3]
- The ancestors of a node are all the nodes along the path from the root to the node.

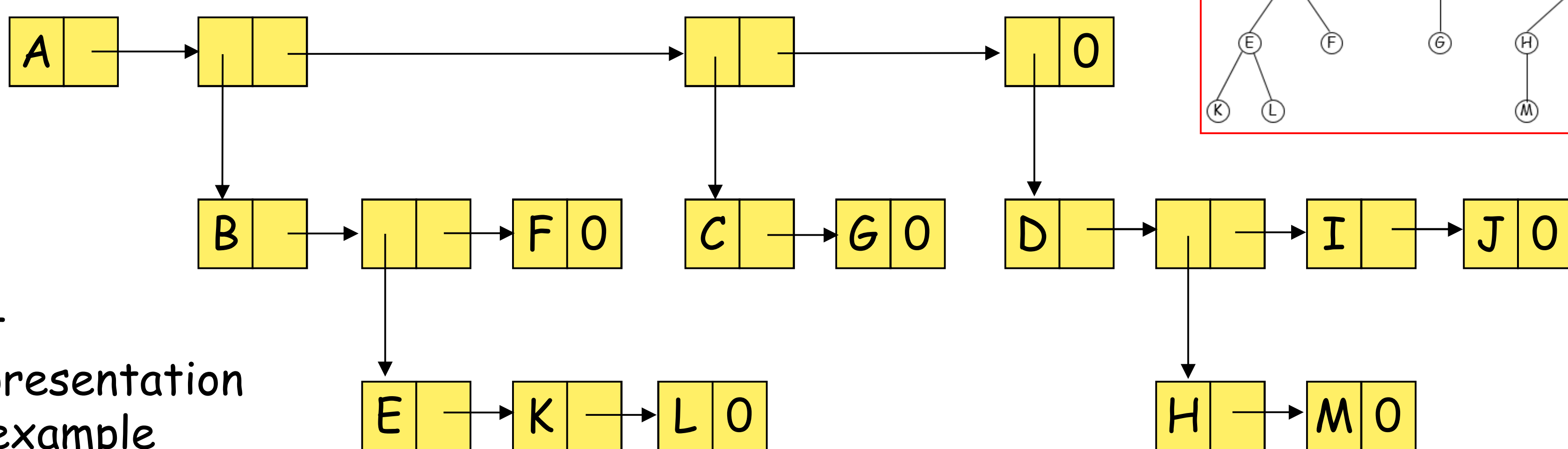
  ancestors of K = A, B and E
- The descendants of a node are all the nodes that are in its subtrees.
- Assume the root is at level 1, then the level of a node is the level of the node's parent plus one.
- The height or the depth of a tree is the maximum level of any node in the tree. [depth of the ex tree = 4]

List Representation of Trees

- Information in root node comes first, followed by a list of the subtrees of that node.
- The example tree could be written as

  (A (B (E (K, L), F), C(G), D(H (M), I, J)))





List Representation of example tree

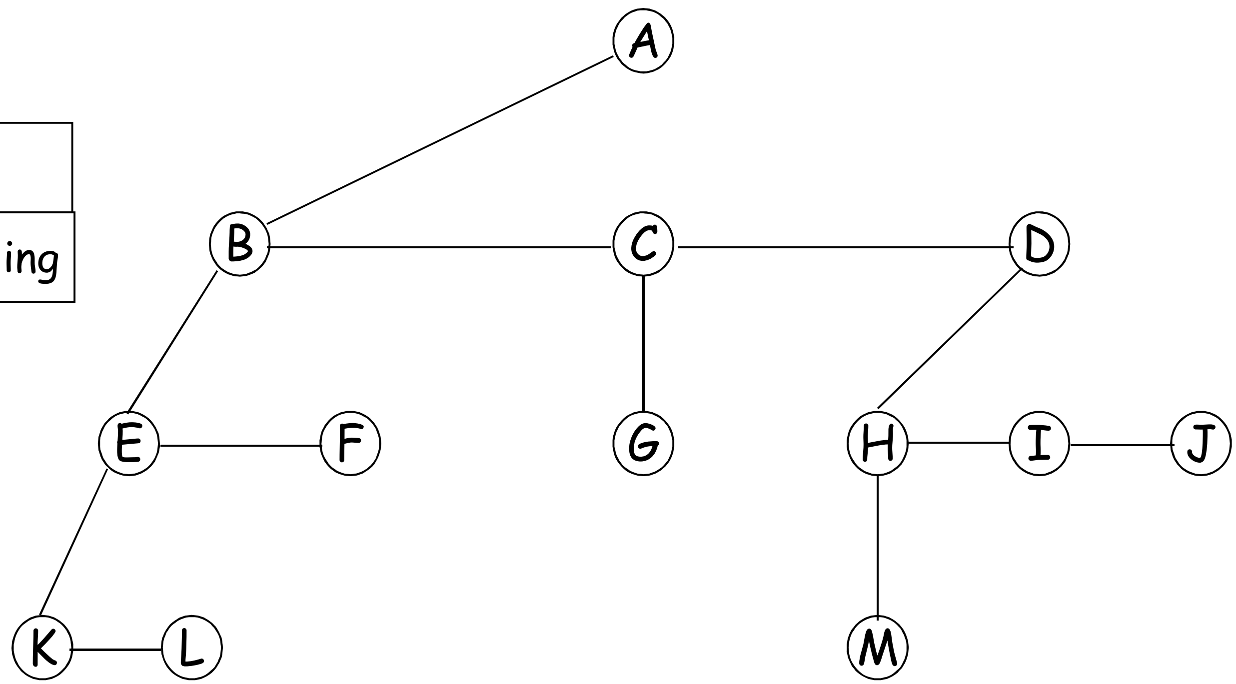
Data	Child1	Chid2	..	Childk

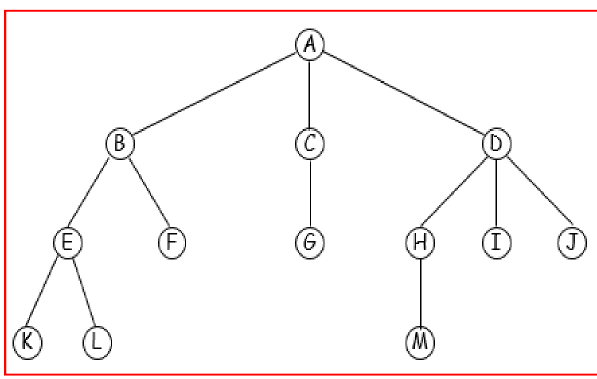
Possible Node Structure for a tree

Representation of Trees

- Left Child-Right Sibling Representation
  - Each node has two links (or pointers).
  - One leftmost child and one closest right sibling.

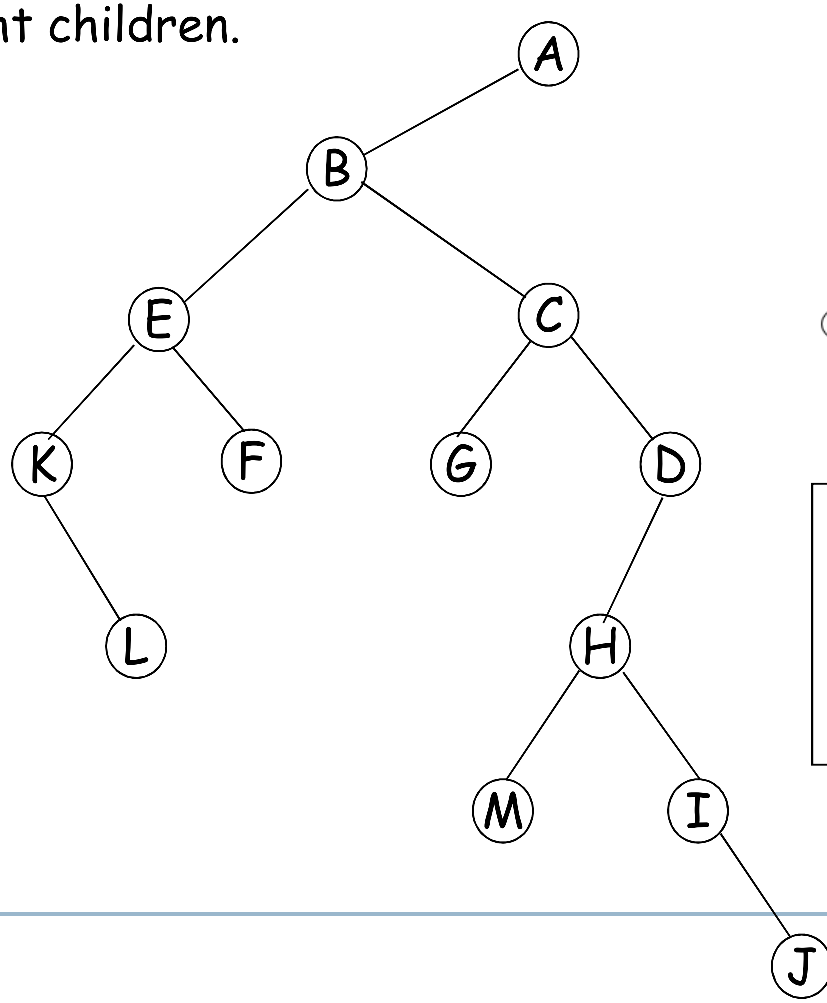
<table>
<tr><td colspan="2">data</td></tr>
<tr><td>left child</td><td>right sibling</td></tr>
</table>

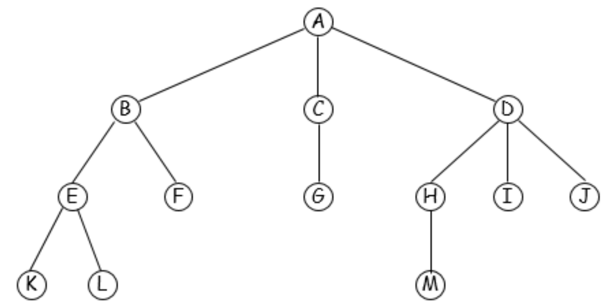




Degree Two Tree Representation

Rotate the right sibling pointers in a left child right sibling tree clockwise by 45 degrees. We refer to the two children of a node as left and right children.

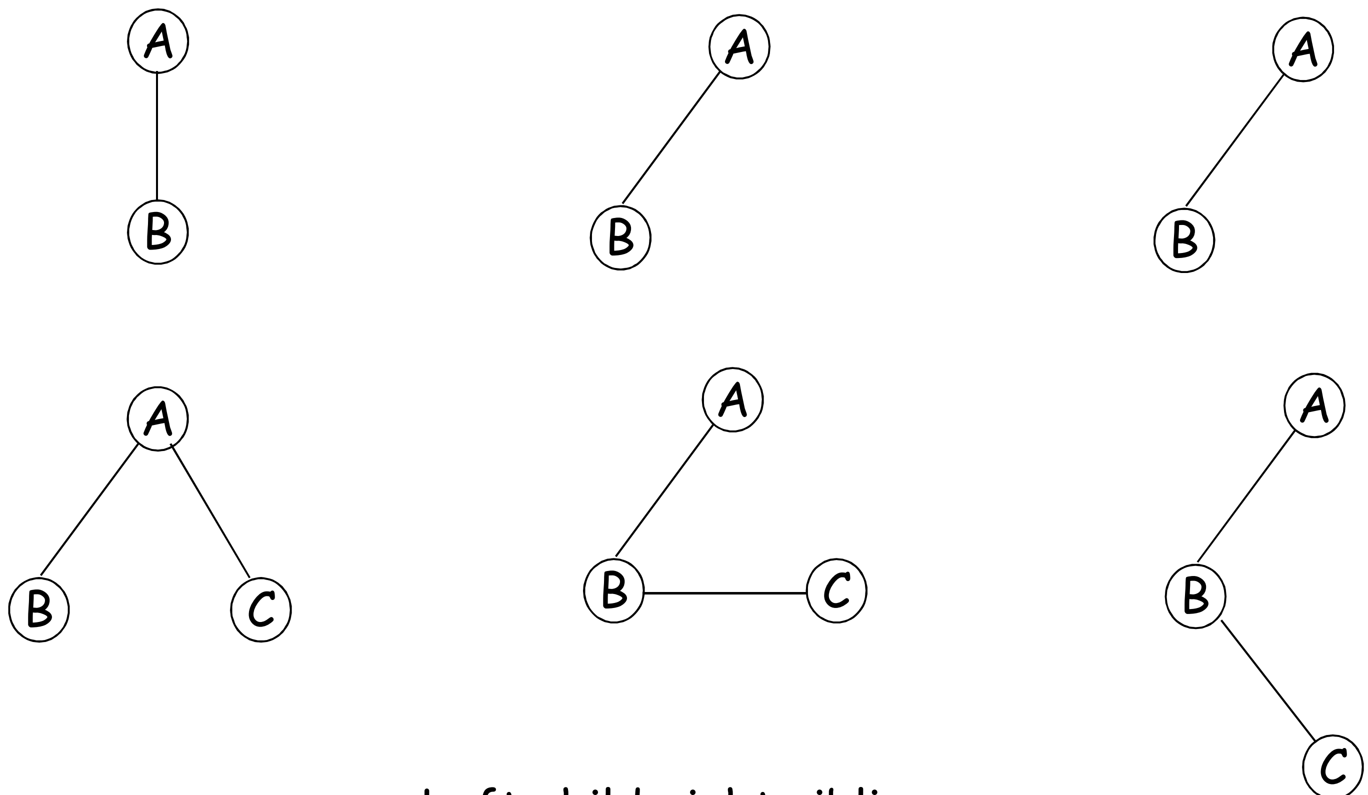




Left Child – Right Child trees are also know as

Binary Trees

Tree Representations



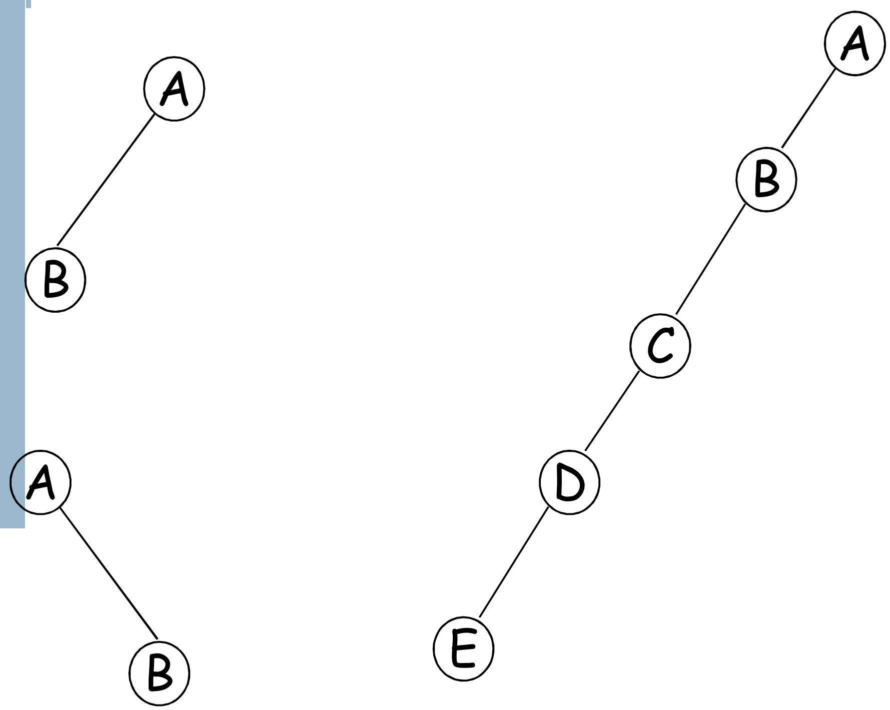
Left child-right sibling

Binary tree

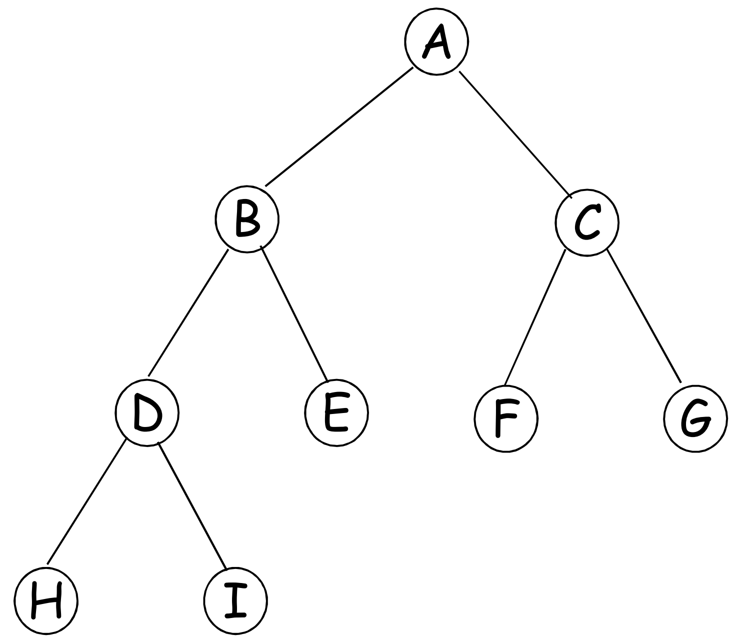
Binary Tree

- Definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree.
- The degree of any given node in a binary tree must not exceed two.
- Binary tree distinguishes between the order of the children while in a tree we do not.

Binary Tree Examples



Skewed Binary Tree



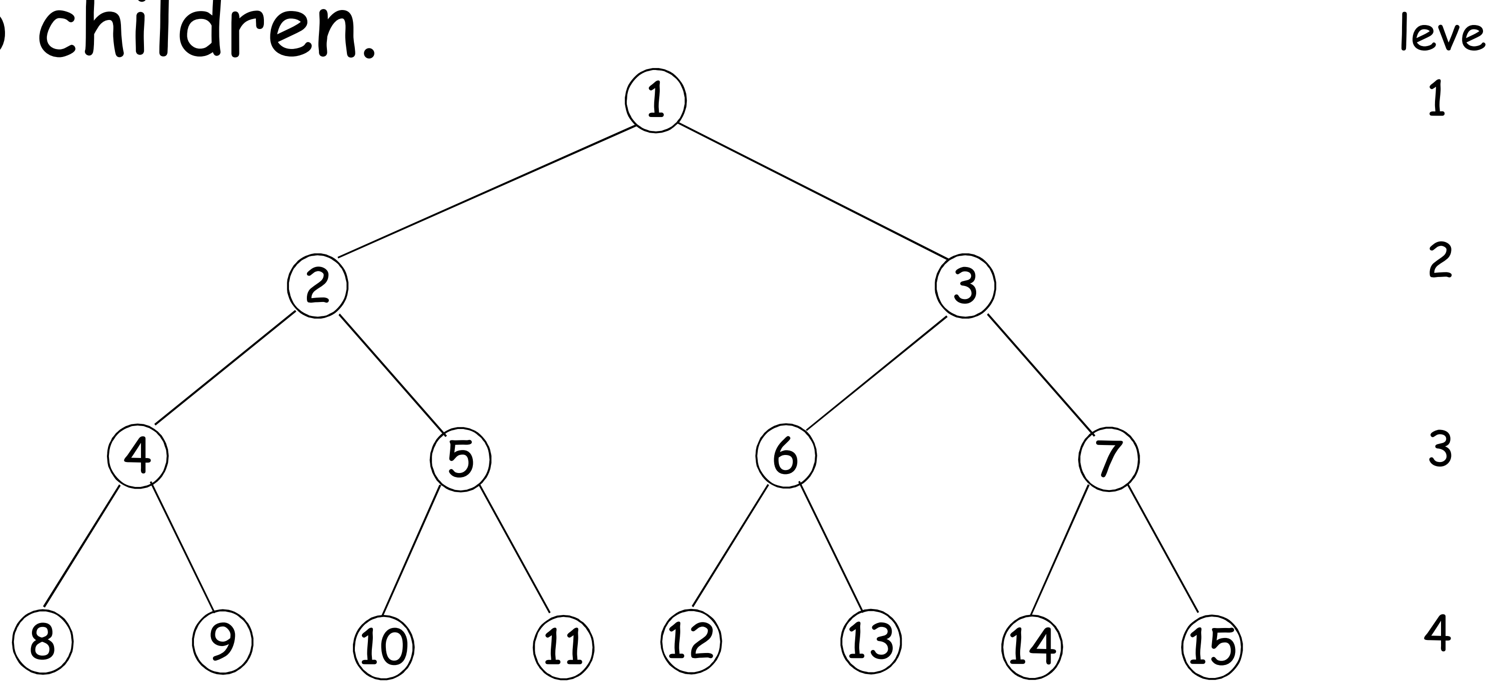
Complete Binary Tree

The Properties of Binary Trees

- **Lemma 5.2** [Maximum number of nodes]
  1) The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i \geq 1$.
  2) The maximum number of nodes in a binary tree of depth k is $2^k - 1$, $k \geq 1$.
- **Lemma 5.3** [Relation between number of leaf nodes and nodes of degree 2]: For any non-empty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes of degree 2, then $n_0 = n_2 + 1$.

Full binary Tree

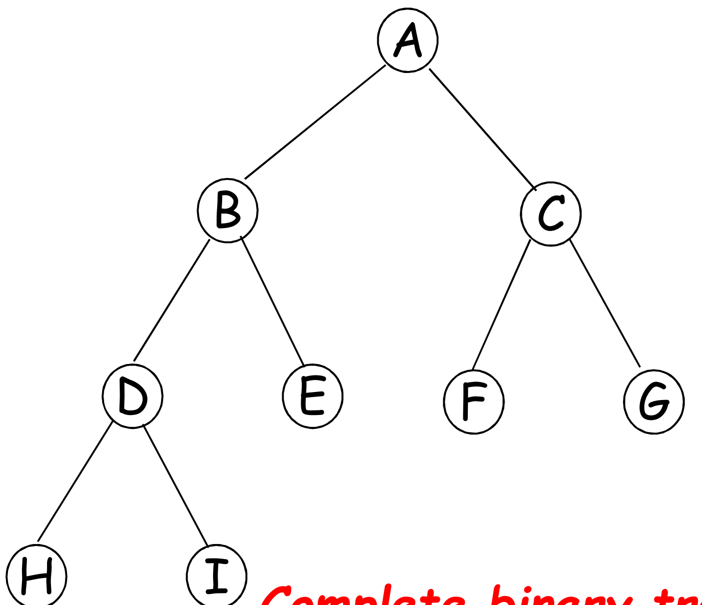
Definition: A **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes, $k \geq 0$ (i.e having the maximum number of nodes). In this every node other than the leaves has two children.



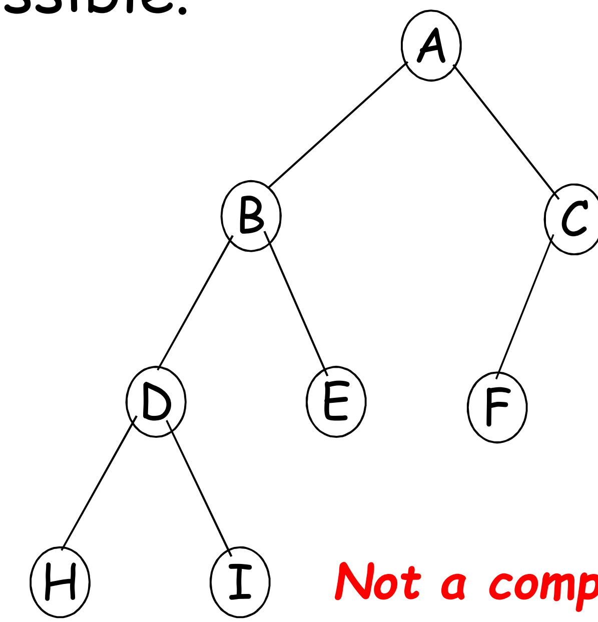
Full Binary Tree of depth 4 with sequential node numbers

Complete binary tree

- **Definition**: A binary tree with n nodes and depth k is ***complete*** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k. In a complete binary tree, every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



Complete binary tree



Not a complete binary tree

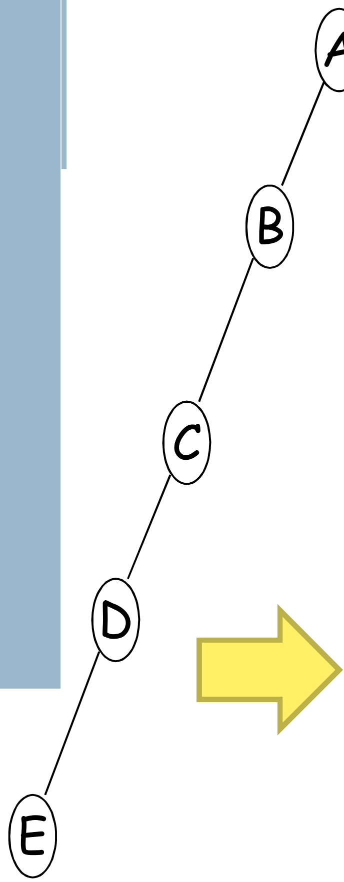
Storage representation of binary trees:

- Trees can be represented using
  - Linear/Sequential (Array) Representation
  - Linked Representation

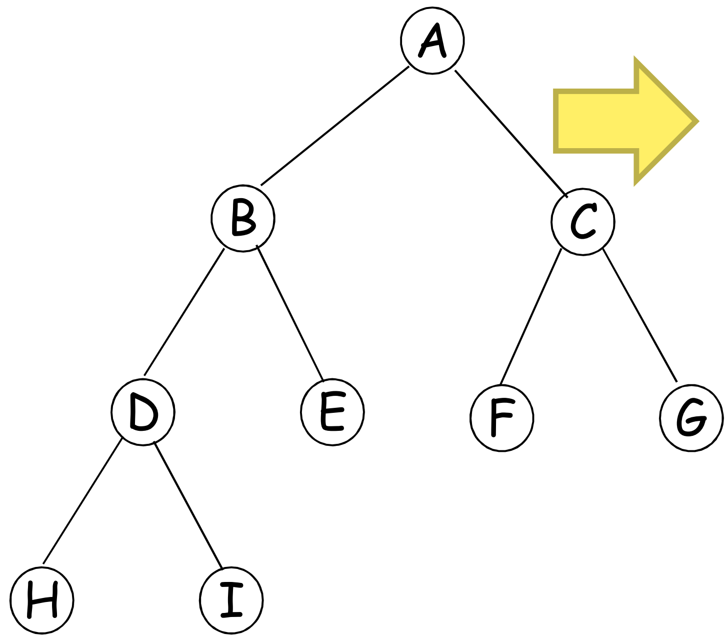
Array Representation of A Binary Tree

- Lemma 5.4: If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \le i \le n$, we have:
  - parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$. If $i = 1$, i is at the root and has no parent.
  - left_child(i) is at $2i$ if $2i \le n$. If 2i > n, then i has no left child.
  - right_child(i) is at $2i + 1$ if $2i + 1 \le n$. If $2i + 1 > n$, then i has no right child.
- Position zero of the array is not used.

Array Representation of Binary Trees



[1]	A
[2]	B
[3]	—
[4]	C
[5]	—
[6]	—
[7]	—
[8]	D
[9]	—
⋮	⋮
[16]	E



[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Advantages and disadvantages of Array representation

Advantages:

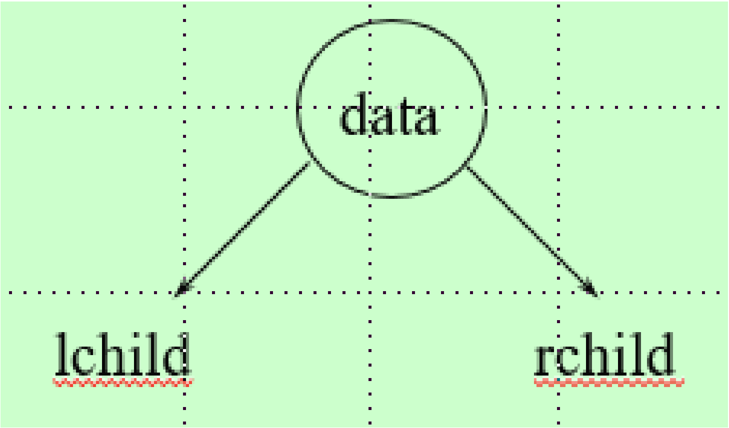
1. This representation is very easy to understand.
2. This is the best representation for full and complete binary tree representation.
3. Programming is very easy.
4. It is very easy to move from a child to its parents and vice versa.

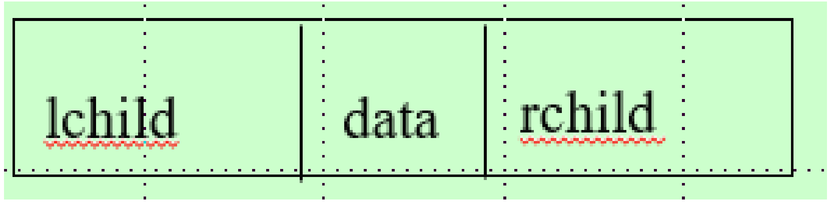
Disadvantages:

1. Lot of memory area wasted.
2. Insertion and deletion of nodes needs lot of data movement.
3. This is not suited for trees other than full and complete tree.

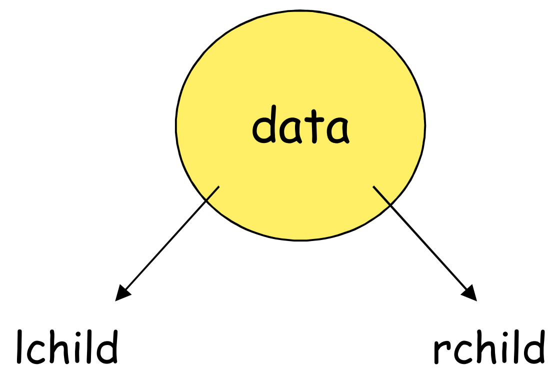
Linked Representation

```
typedef struct node *Nodeptr;

struct node{
    int data;
    Nodeptr rchild;
    Nodeptr lchild;
};
```

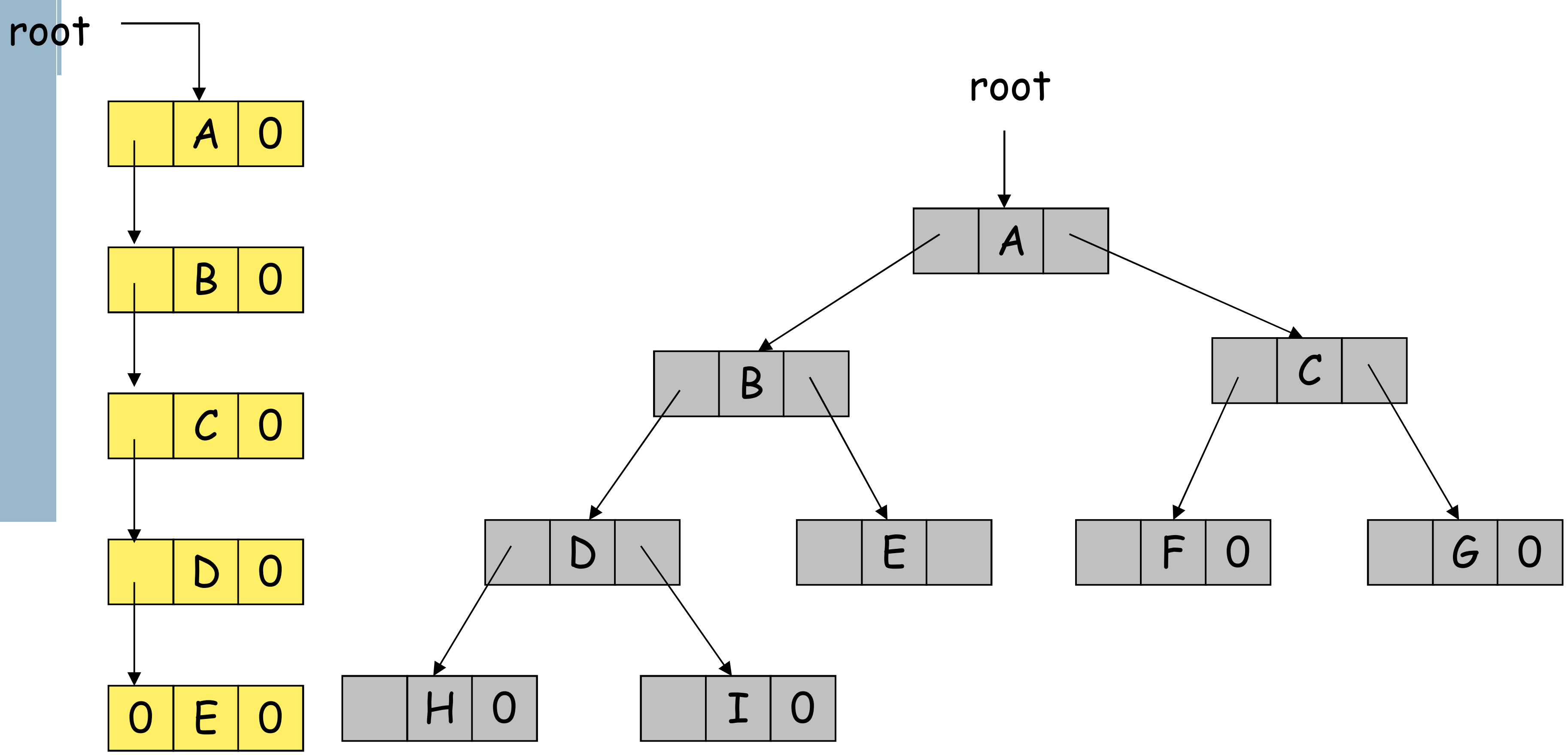




Node Representation



Linked List Representation For The Binary Trees



Advantages and disadvantages of linked representation

Advantages

1. A particular node can be placed at any location in the memory.
2. Insertions and deletions can be made directly without data movements.
3. It is best for any type of trees.
4. It is flexible because the system take care of allocating and freeing of nodes.

Disadvantage

1. It is difficult to understand.
2. Additional memory is needed for storing pointers
3. Accessing a particular node is not easy.

Recursive Function to create a binary tree

```
Nodeptr CreateBinaryTree(int item){
    int x;

    if (item!=-1)  { //until input is not equal to -1
            Nodeptr temp = getnode();
            temp->data = item;

            printf("Enter the lchild of %d :",item);
            scanf("%d",&x);
            temp->lchild = CreateBinaryTree(x);

            printf("Enter the rchild of %d :",item);
            scanf("%d",&x);
            temp->rchild = CreateBinaryTree(x);

            return temp;
    }
    return NULL;
}
```

```
int main()
{
  Nodeptr root = NULL;
   int item;

   printf("Creating the tree : \n");
   printf("Enter the root : ");
   scanf("%d",&item);

   root=CreateBinaryTree(item);
        ...
```

Tree Traversal

- Let L, V, and R stand for moving left, visiting the node, and moving right.
- There are six possible combinations of traversal for a binary tree
  - LVR, LRV, VLR, VRL, RVL, RLV
- Adopt convention that we traverse left before right, only 3 traversals remain
  - LVR, LRV, VLR
  - →inorder, postorder, preorder
- When implementing the traversal, a recursion is perfect for the task.

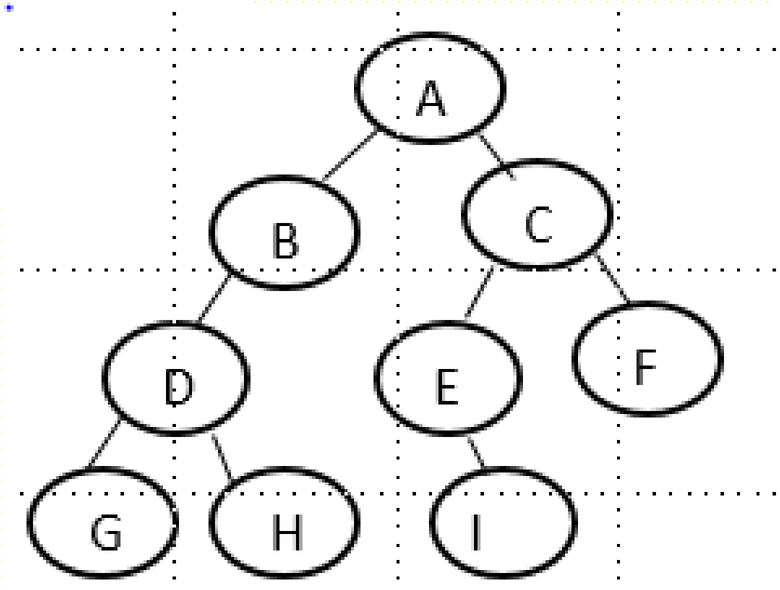
Tree Traversal

<u>Inorder traversal</u>

- It can be recursively defined as follows.
  1. Traverse the left subtree in inorder.
  2. Process the root node.
  3. Traverse the right subtree in inorder.

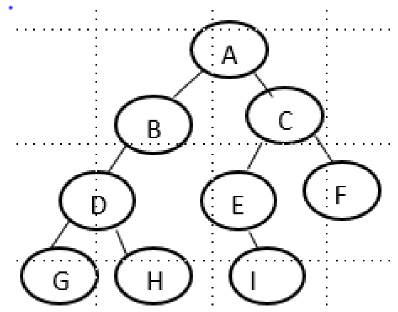
Inorder Traversal

- Move towards the left of the tree(till the leaf node), display that node and then move towards right and repeat the process.
- Since same process is repeated at every stage, recursion will serve the purpose.
- Example:

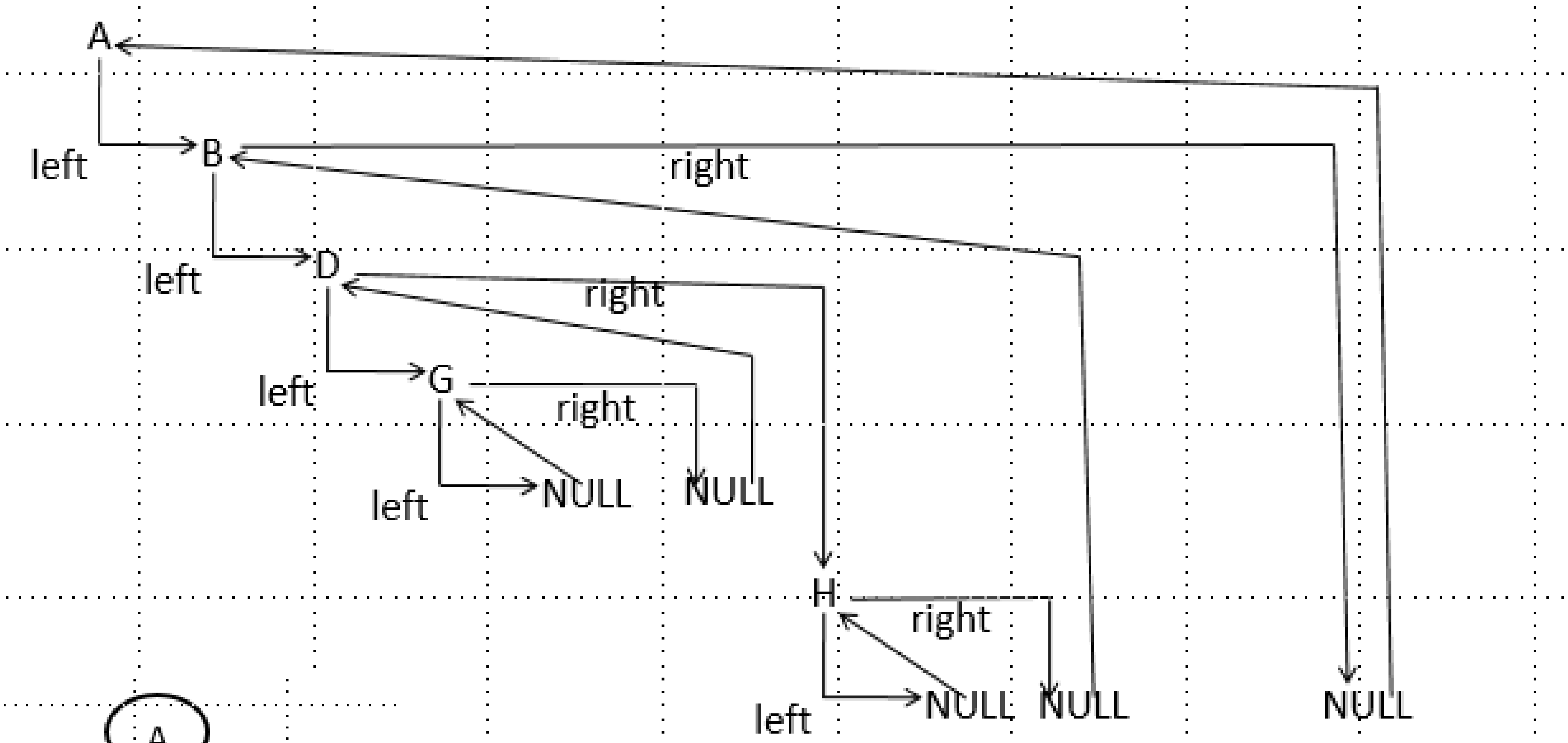


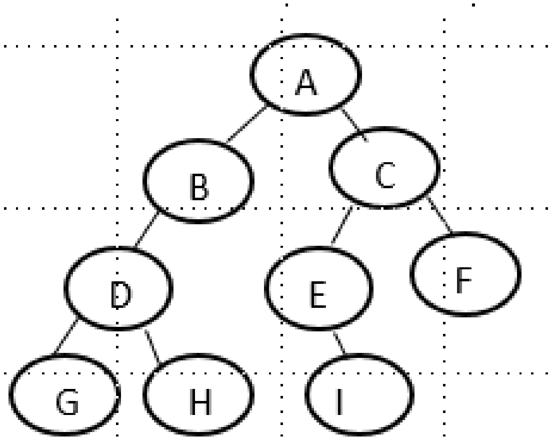
Inorder traversal of above tree gives GDHBAEICF

Inorder Traversal - Example

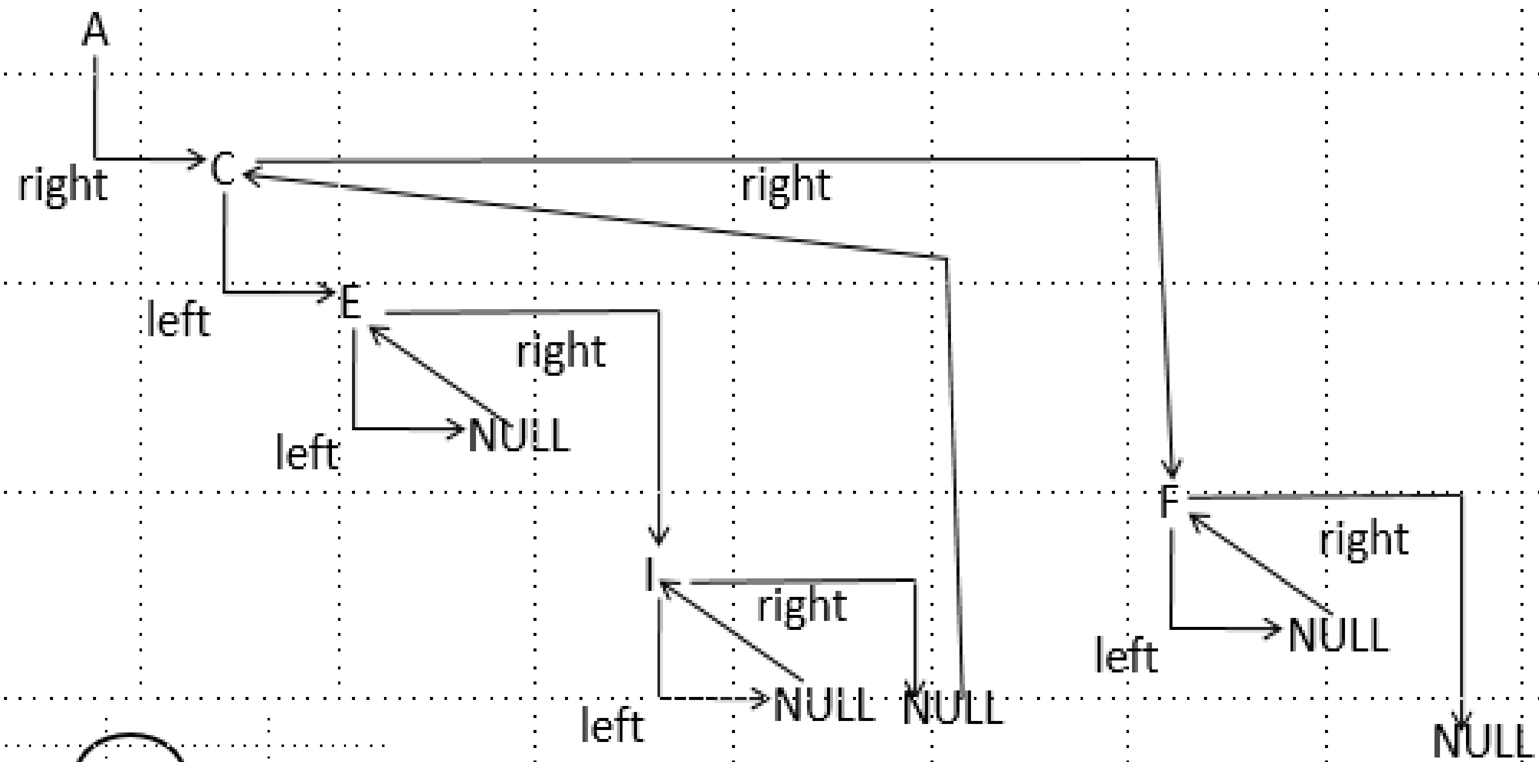


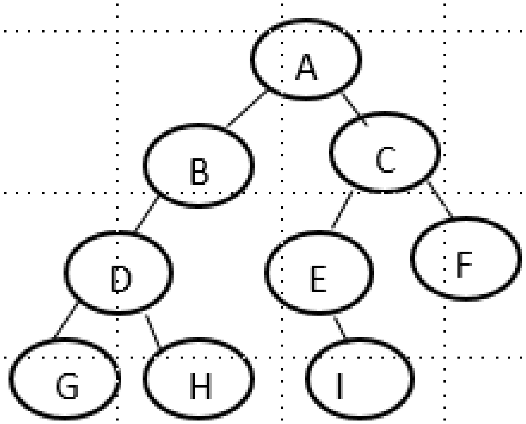
- Move towards left, we end up in G. G does not have a left child. Now display the root node(in this case it is G). Hence G is displayed first.
- Move to the right of G, which is also NULL. Hence go back to root of G and print it. So D is printed next.
- Go to the right of D, which is H. Now another root H is visited.
- Move to the left of H, which is NULL. So go back to root H and print it and go to right of H, which is NULL.
- Go back to the root B and print it and go right of B, which is NULL. So go back to root of B, which is A and print it.
- Traversing of left subtree is finished and so move towards right of it & reach C.
- Move to the left of C and reach E. Again move to left, which is NULL. Print root E and go to right of E to reach I.
- Move to left of I, which is NULL. Hence go back to root I, print it and move to its right, which is NULL.
- Go back to root C, print it and go to its right and reach F.
- Move to left of F, which is NULL. Hence go back to F, print it and go to its right, which is also NULL.


A
left
B
right
left
D
right
left
G
right
left
NULL
NULL
H
right
left
NULL
NULL
NULL


A
B
C
D
E
F
G
H
I

G D H B A


A
right
C
right
left
E
right
left
NULL
I
right
left
NULL
NULL
F
right
left
NULL
NULL

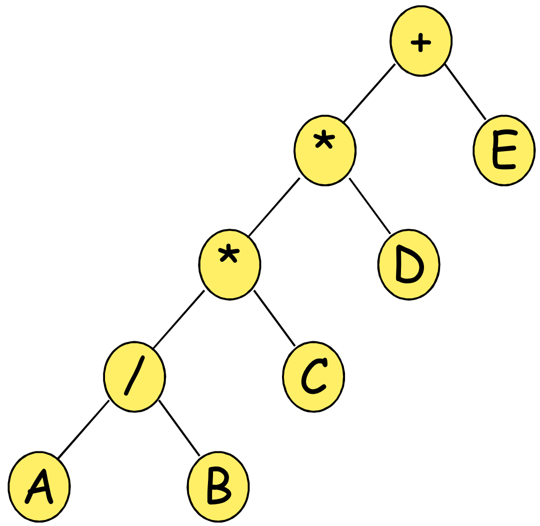

A
B
C
D
E
F
G
H
I

E I C F

Inorder Traversal

```
/*recursive algorithm for inorder traversal*/

void inorder(Nodeptr root)
{
   if (root)
   {
         inorder(root→lchild);
         printf("%d ", root→data);
         inorder(root→rchild);
   }
}
```

Inorder Traversal Example



Binary tree with operators and operands - Expression tree

Call of *inorder*	Value in *root*	Action	*inorder*	in *root*	Value Action
1	+		11	*C*	
2	*		12	*NULL*	
3	*		11	*C*	**printf**
4	/		13	*NULL*	
5	*A*		2	*	**printf**
6	*NULL*		14	*D*	
5	*A*	**printf**	15	*NULL*	
7	*NULL*		14	*D*	**printf**
4	/	**printf**	16	*NULL*	
8	*B*		1	+	**printf**
9	*NULL*		17	*E*	
8	*B*	**printf**	18	*NULL*	
10	*NULL*		17	*E*	**printf**
3	*	**printf**	19	*NULL*	

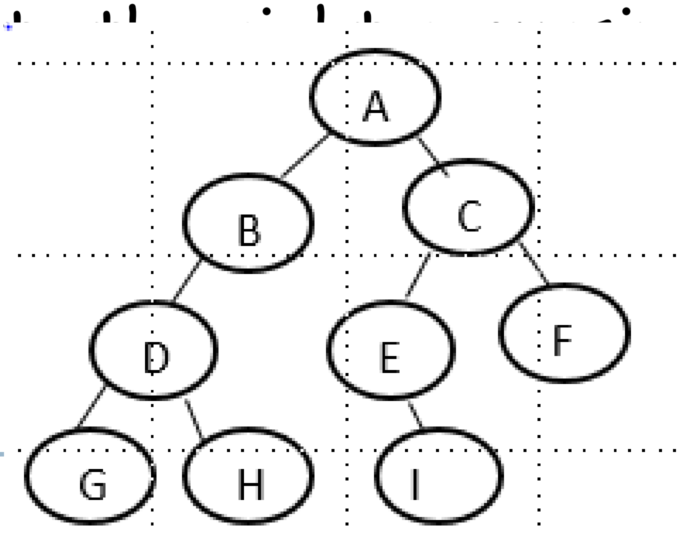
Trace of the program

Preorder Traversal

Preorder traversal is defined as

1. Process the node.
2. Traverse the left subtree in preorder.
3. Traverse the right subtree in preorder.

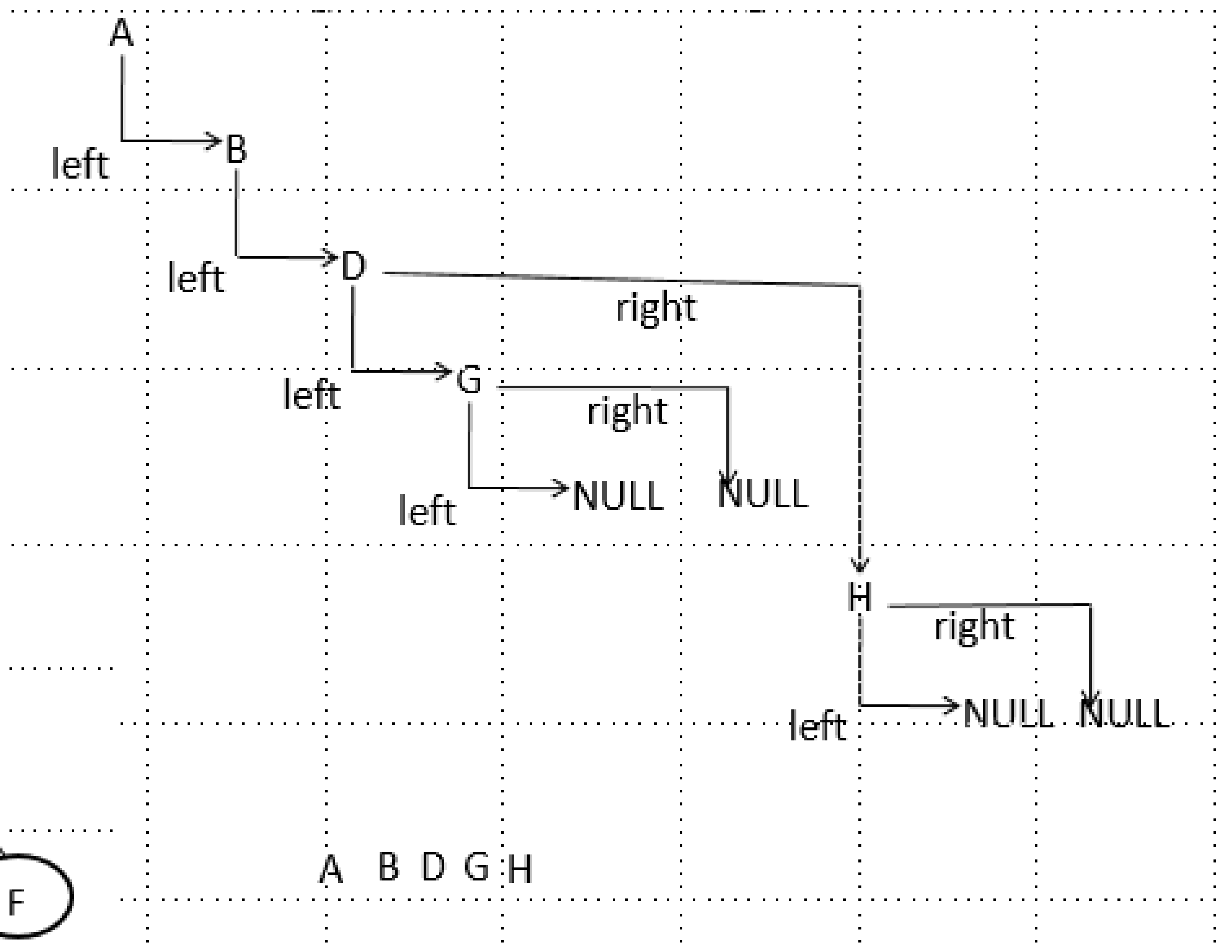
- In preorder, we first visit the node, then move towards left and then [illegible]ely.

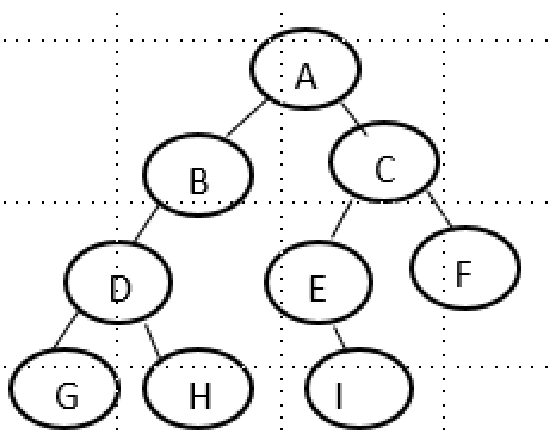


Preorder Traversal

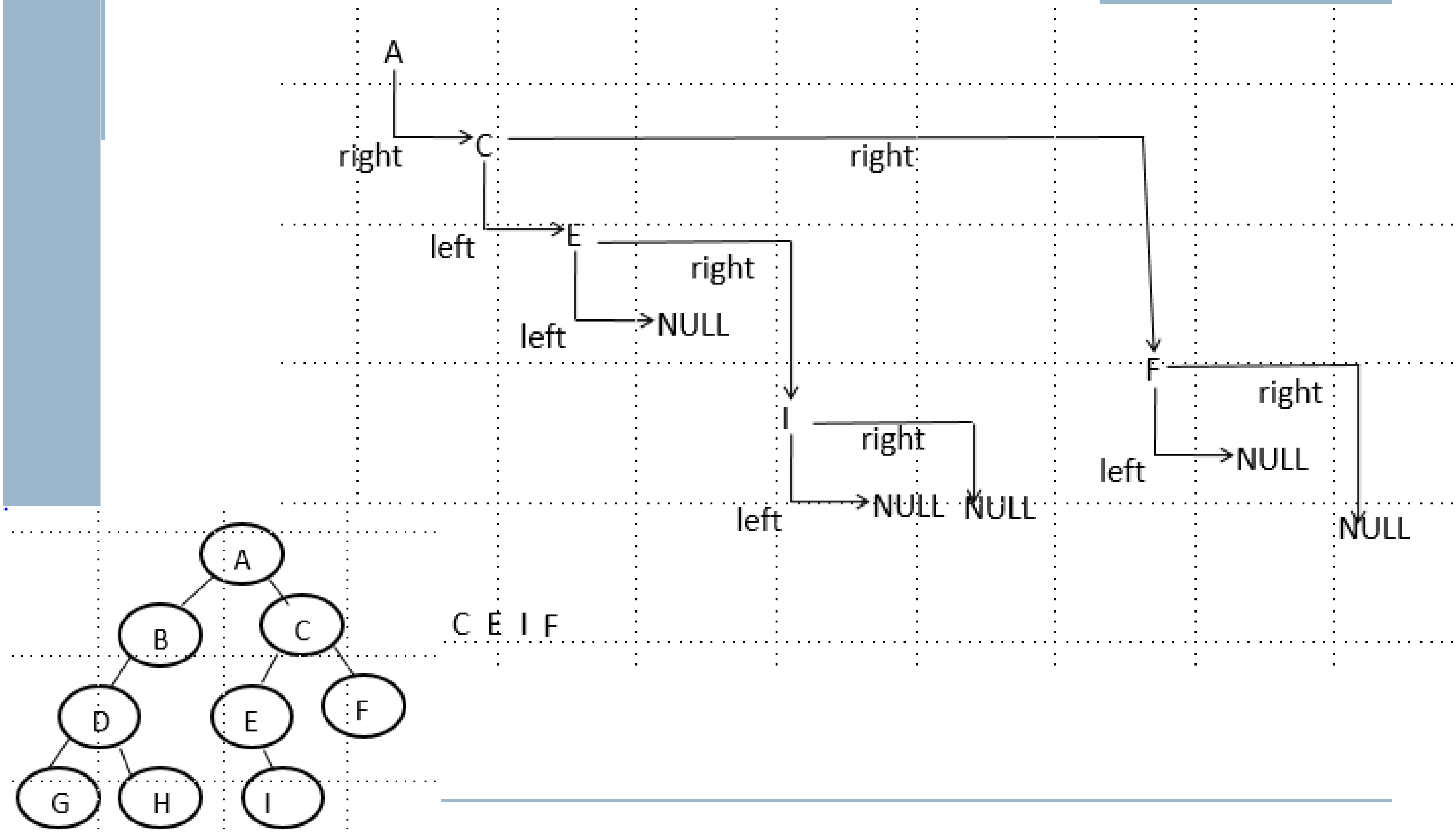
```
void preorder(Nodeptr root)
{

   if(root)
   {
         printf("%d ", root→data);
         preorder(root→lchild);
         preorder(root→rchild);

   }
}
```

Traversing left sub tree in preorder





Traversing right sub tree in preorder



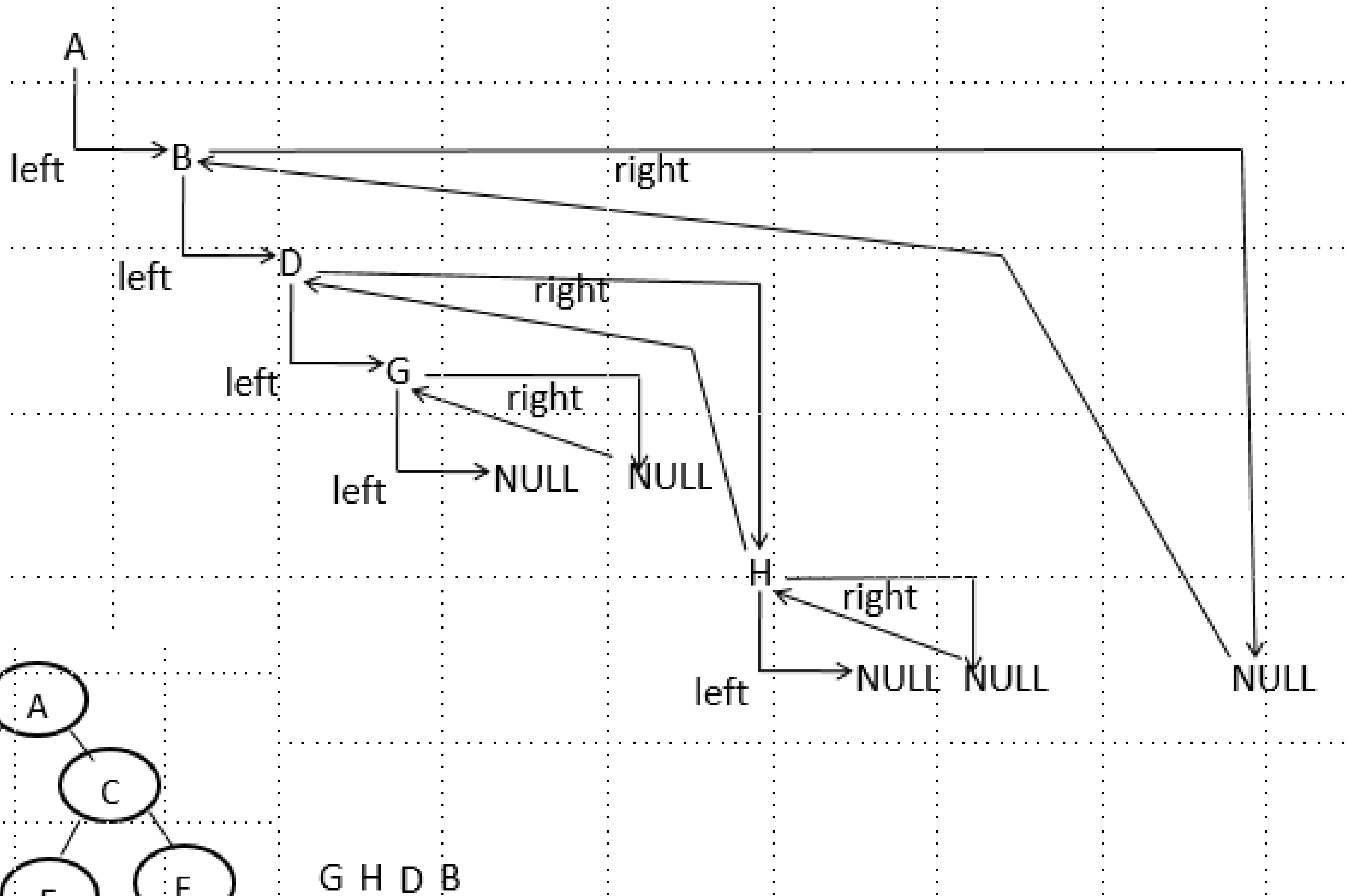
Post order traversal

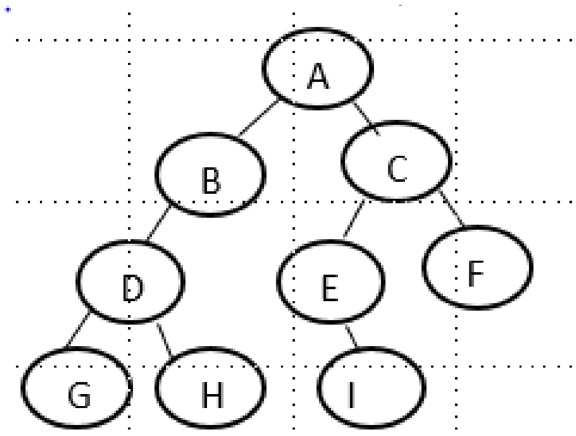
Post order traversal is defined as

1. Traverse the left subtree in postorder.
2. Traverse the right subtree in postorder.
3. Process the root node.

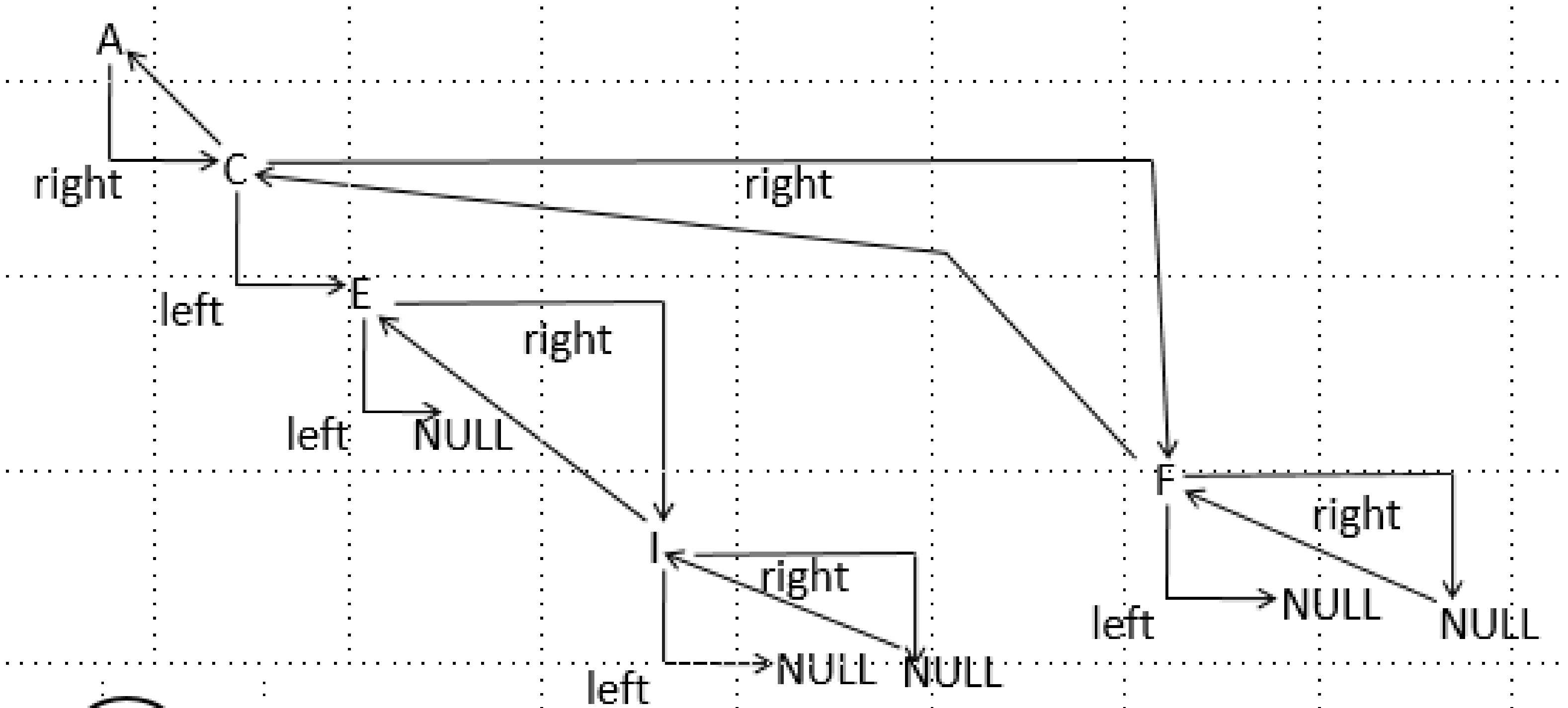
- In post order traversal, we first traverse towards left, then move to right and then visit the root. This process is repeated recursively.

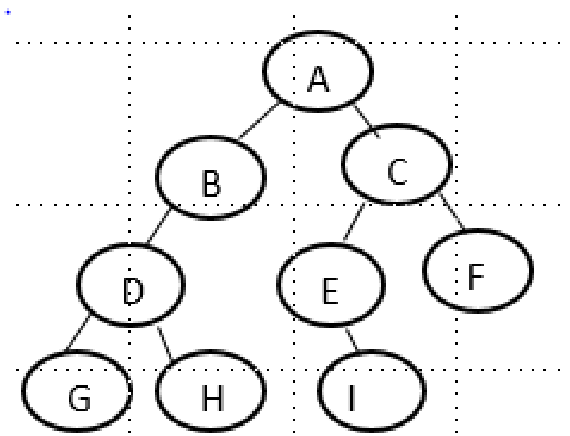
Post order traversal-Example





Post order traversal - Example



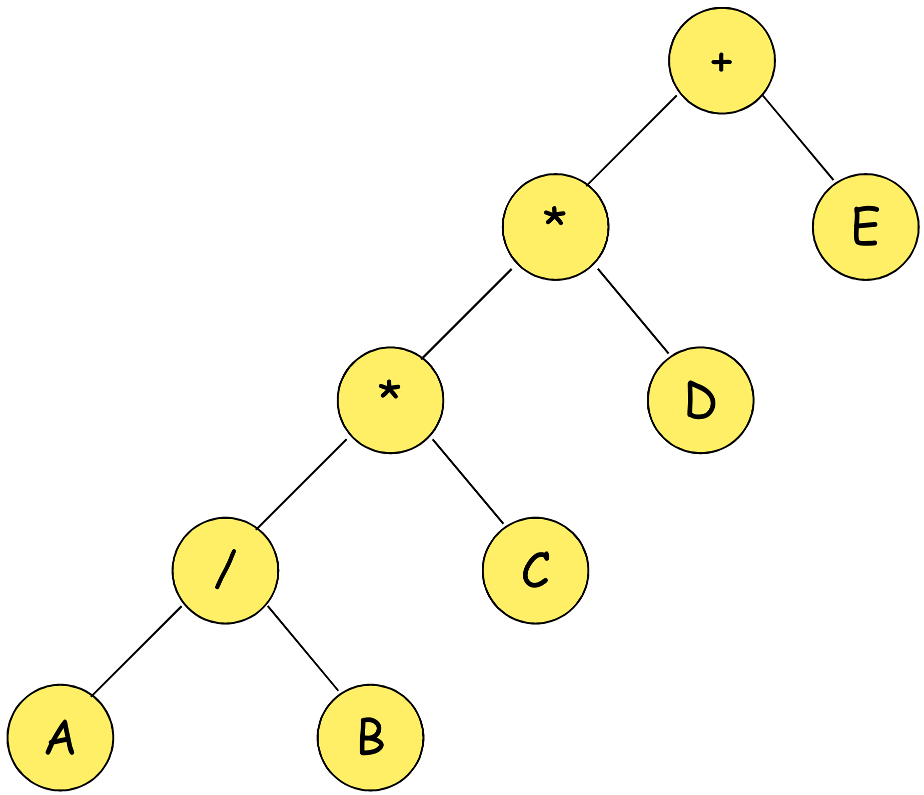


I E F C A

Post order traversal

```
void postorder(Nodeptr root)
{
        if(root)
        {
                postorder(root→lchild);
                postorder(root→rchild);
                printf("%d ",root→data);
        }
}
```

Binary Tree With Arithmetic Expression



Tree Traversal

- Inorder Traversal: A/B*C*D+E
  => Infix form
- Preorder Traversal: +**/ABCDE
  => Prefix form
- Postorder Traversal: AB/C*D*E+
  => Postfix form