CSE 2103
(Data Structures)

Lecture on
Chapter-7: Tree

By

Dr. M. Golam Rashed

(golamrashed@ru.ac.bd)

Department of Computer Science and Engineering(CSE)
Varendra University, Rajshahi, Bangladesh

Trees and Graphs: Basic terminology, Binary trees, Binary tree representation, Tree traversal algorithms, Extended binary tree, Huffman codes/algorithm, Graphs, Graph representation, Shortest path algorithm and transitive closure, Traversing a graph.

- Linear Data Structures
  - ✓ Strings
  - ✓ Arrays
  - ✓ Linked Lists
  - ✓ Stacks, and
  - ✓ Queues
- Nonlinear Data Structures
  - ✓ Tree,
  - ✓ Graphs

Tree Data Structure

- Tree structure is mainly used to represent data containing a hierarchical relationship between elements. For example….
  - Records,
  - Family tress, and
  - Tables of contents.
- In this chapter, we investigate a special kind of tree, called a ***binary tree***
  - It can be maintained in the computer easily

Binary Tree

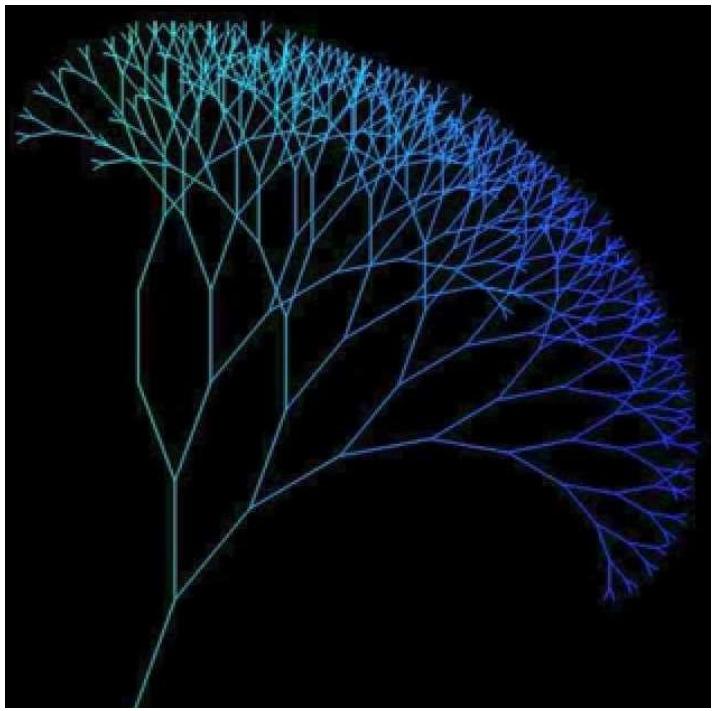
- T is a finite set of elements , called *Nodes,* such that
  - T is empty (called null tree or empty tree), or
  - T contains a distinguished node R, called a root of T, and the remaining nodes form an ordered pair of disjoint binary trees T1, and T2
- Any node of T has 0, 1 , or 2 children.

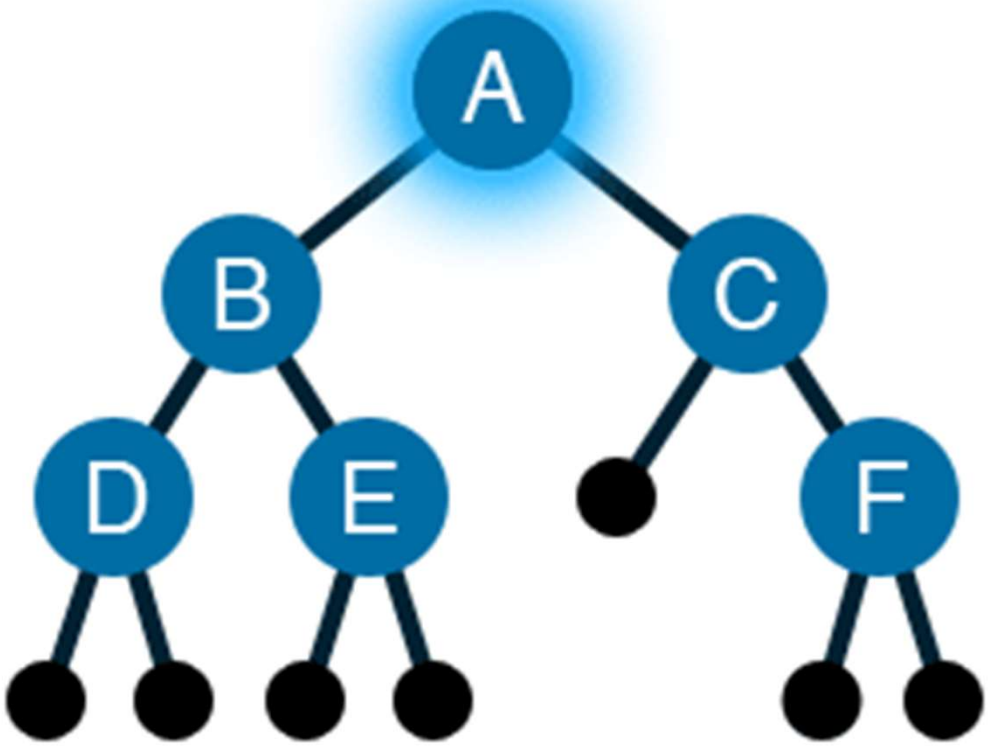
Basic Binary Tree Concept

- If T does contain a Root R , then the two trees T_1 and T_2 are called, respectively , the left and right sub-trees of R.
- If T_1 is nonempty, then its root is called the left successor of R; similarly,
- If T_2 is nonempty, then its root is called the right successor of R;

Binary Tree Illustration

- A Binary Tree T is frequently presented by mean of a diagram





Output:

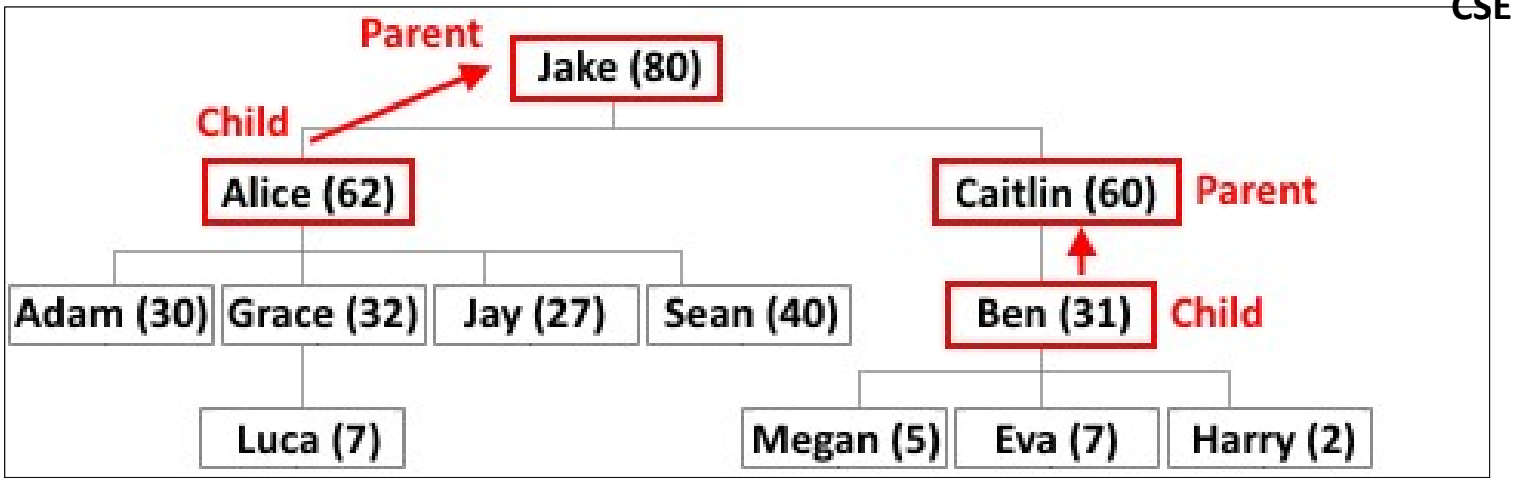
See Example 7.1 and 7.2 for understanding

Tree Terminology

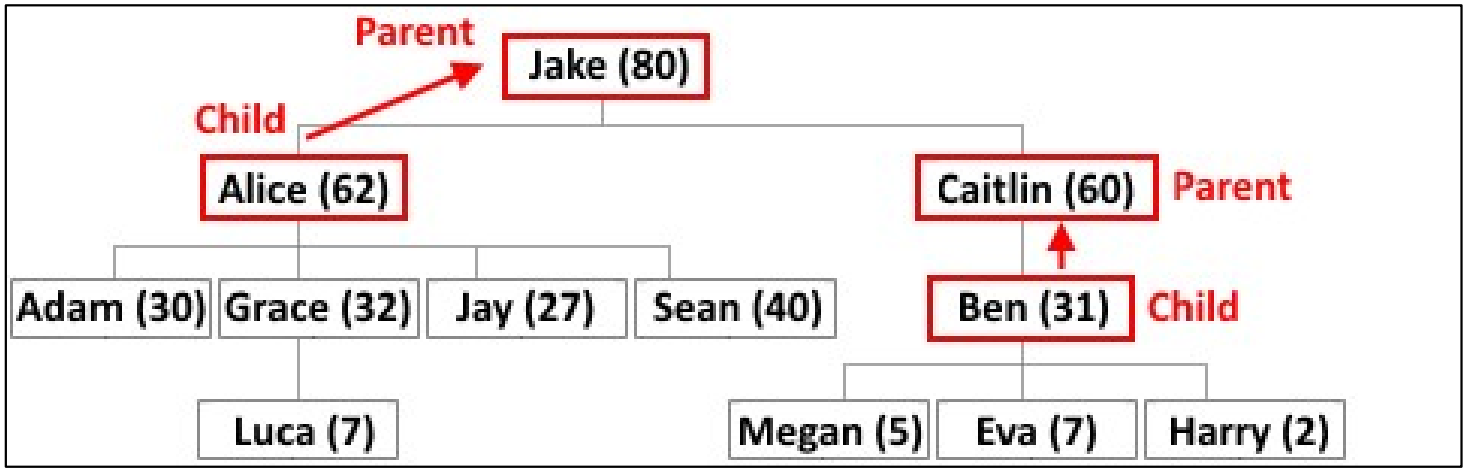
- Suppose N is a node in T with left successor S1 and right successor S2.
  - N is called the parent (or father) of S1 and S2.
- Analogously, S1 is called the left child (or son) of N, and S2 is called the right child (or son) of N
- S1 and S2 are said to be siblings (or brothers)
- Two or more nodes with the same parents are called siblings.

Tree Terminology

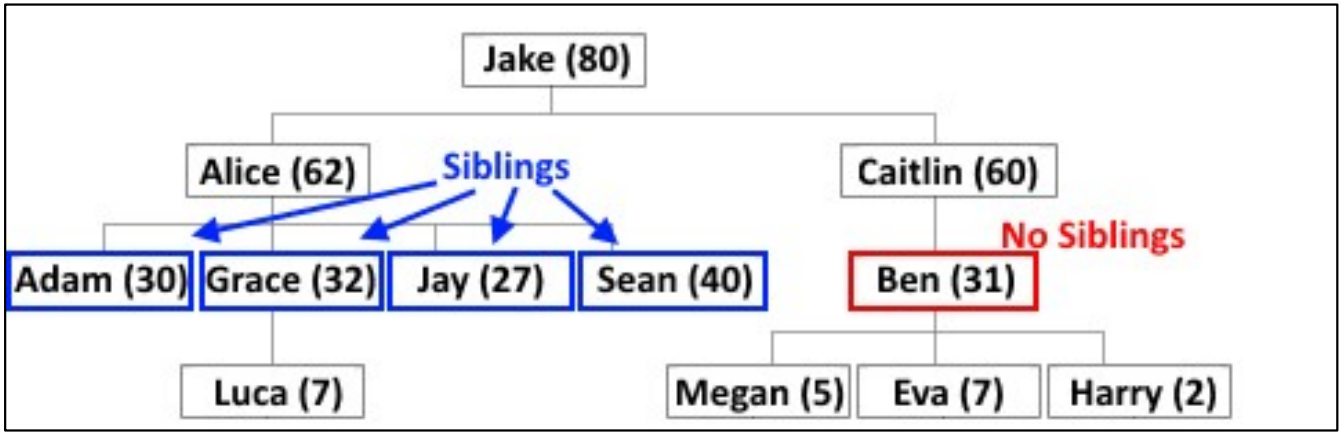
Parent



Child



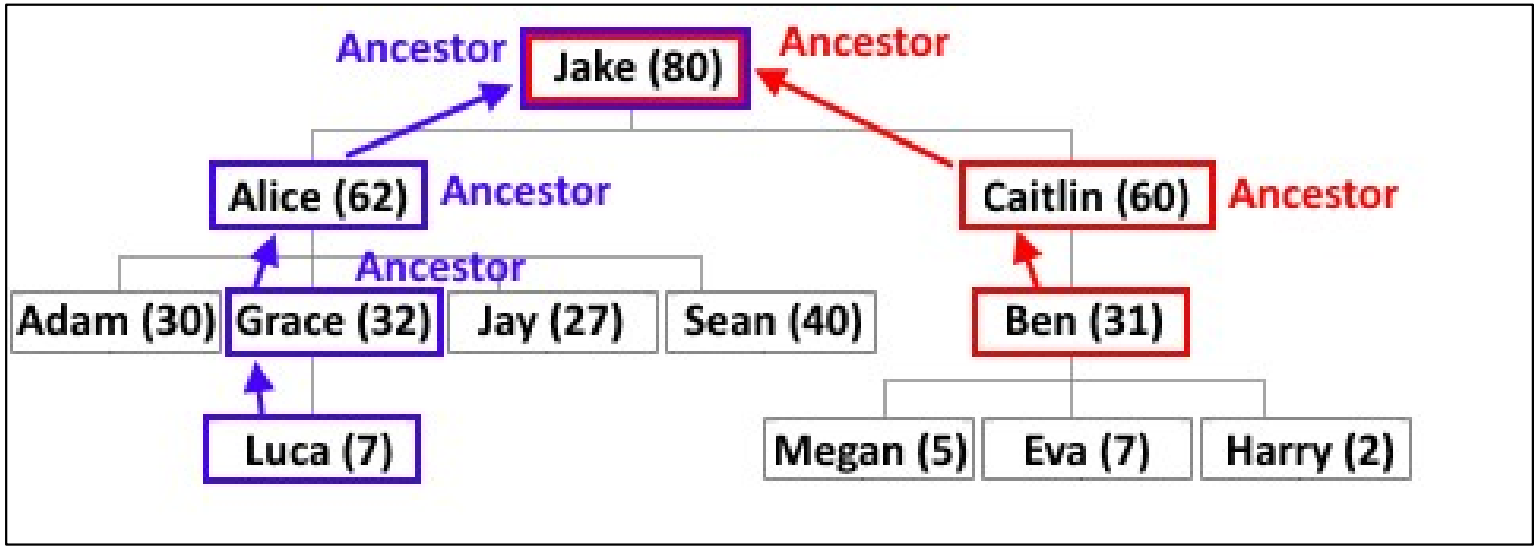
Sibling



Tree Terminology

- An ancestor is any node in the path from the root to the node.

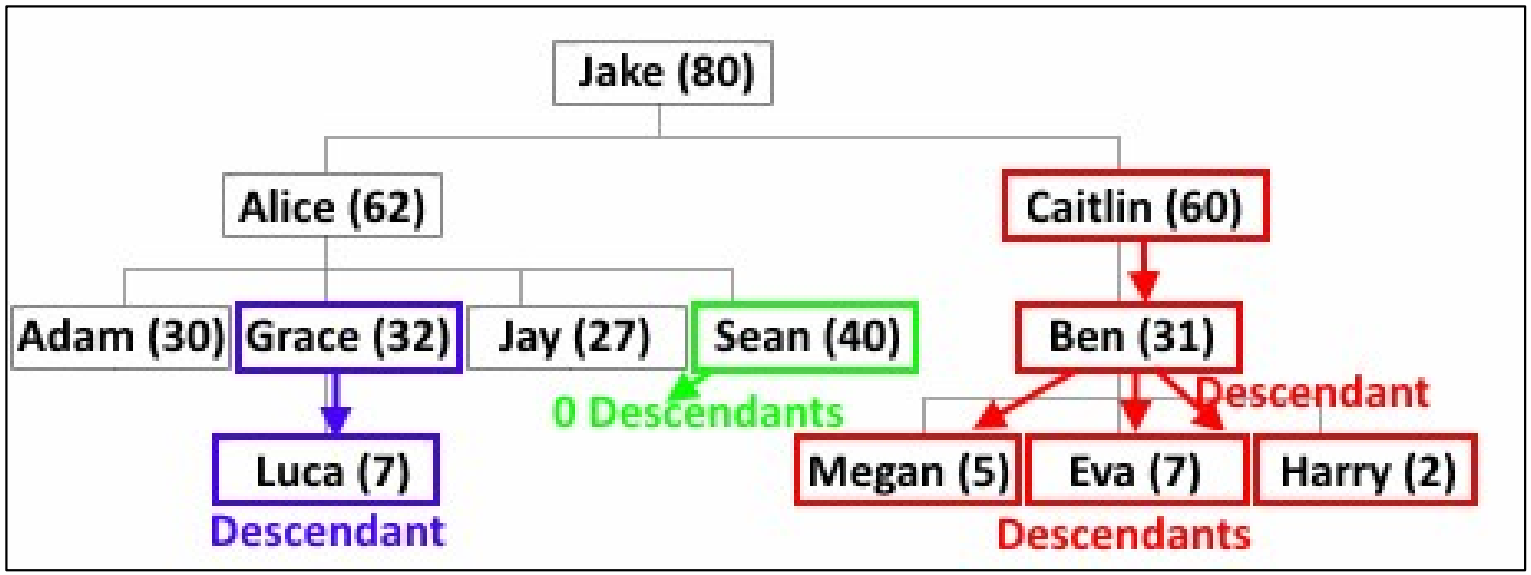
Ancestor



Tree Terminology

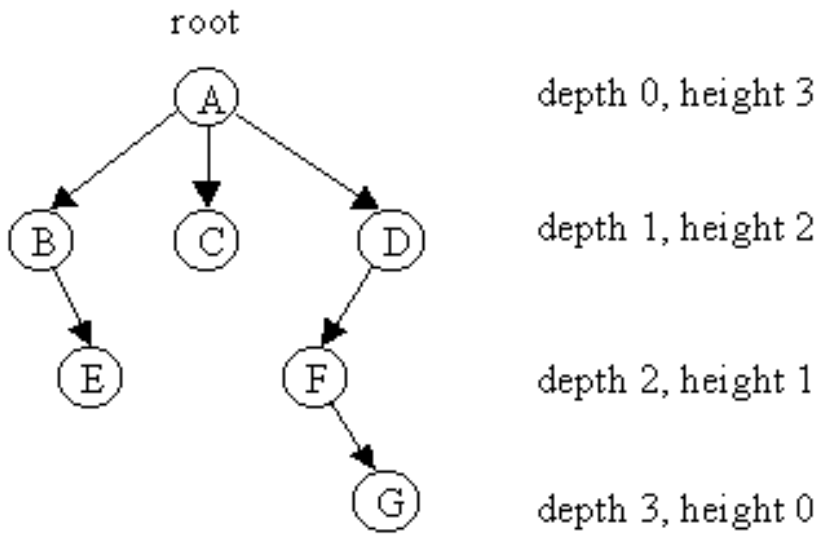
- A descendant is any node in the path below the parent node; that is, all nodes in the paths from a given node to a leaf are descendants of that node.

Descendant



Tree Terminology

- A path is a sequence of nodes in which each node is adjacent to the next node.
- The level or depth of a node is its distance from the root. The root is at level 0, its children are at level 1, etc.
- The height of the tree is the level of the leaf in the longest path from the root.

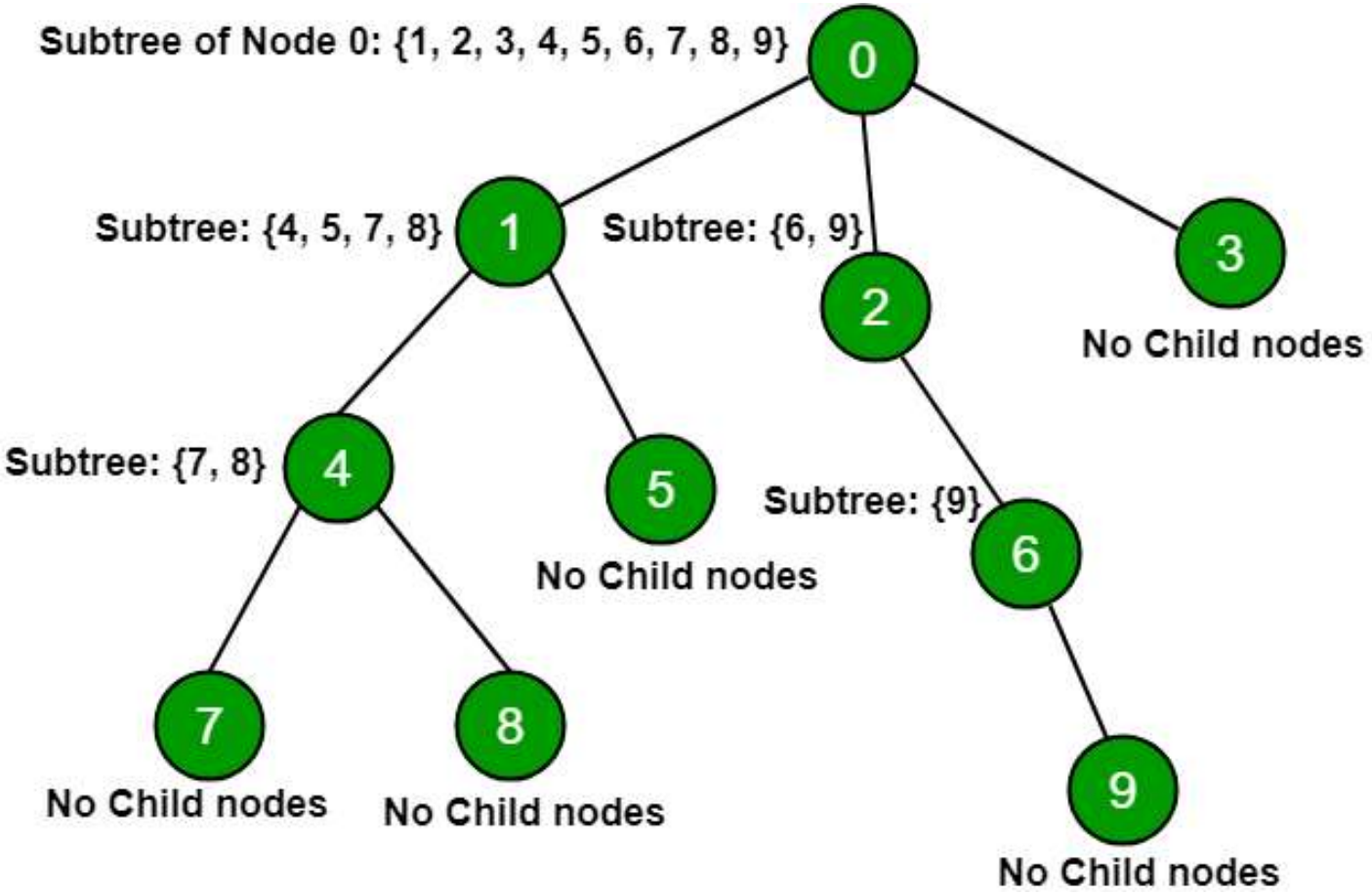


A tree of height 3

Tree Terminology

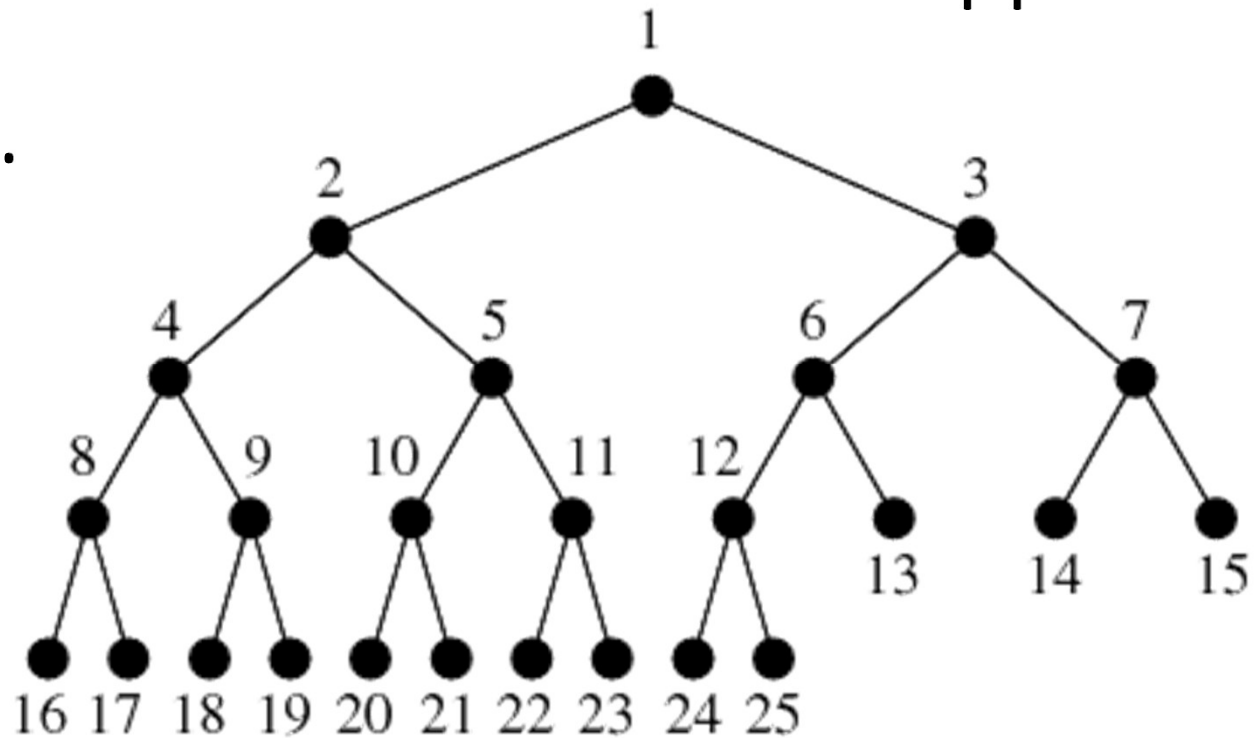
- A subtree is any connected structure below the root.

- The first node in the subtree is known is the root of the subtree.



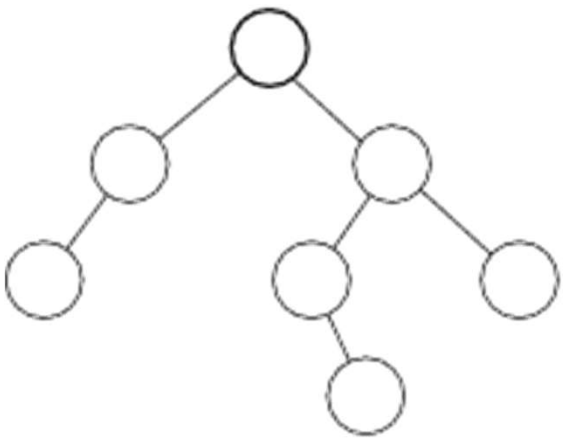
Complete Tree

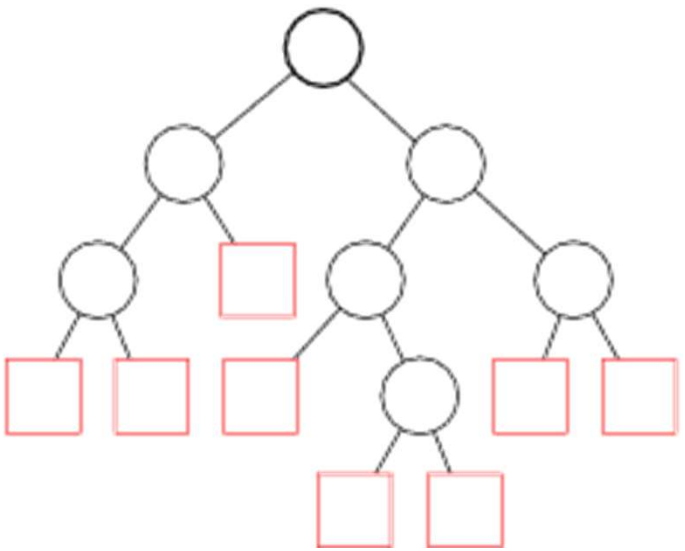
- A tree is said to be **complete if all its levels, except possibly the last have the maximum number of possible nodes**, and
- if all the nodes at the last level appear as far left as possible.



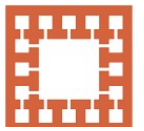
Extended Binary Tree: 2-Tree

- A binary tree T is said to be a 2-tree or an extended binary tree if each node N has either 0 or 2 children.
- In such a case, the nodes, with 2 children are called internal nodes, and the node with 0 children are called external node.





Representing Binary Tree in Memory

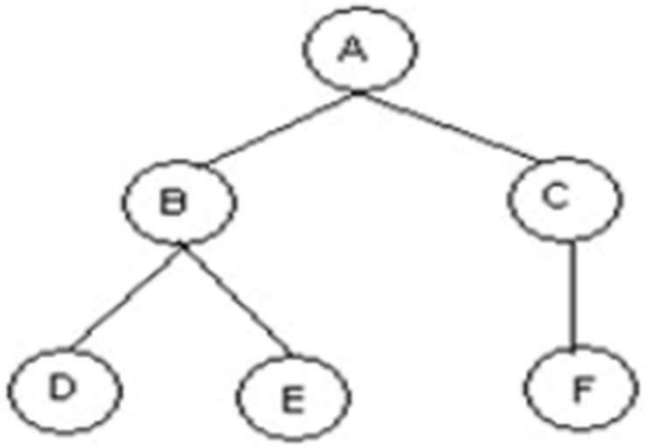


- Let T be a binary tree.
- There are two ways of representing T in memory:
  - First and usual was is called link representation of T
  - The second way uses a single array, called the sequential representation of T
- The main requirement of any representation of T is that one should have direct access to the root R of T and, given any node N of T, one should have direct access to the children of N.

Linked Representation of Binary Trees

Linked Representation of Binary Tree

- Use Three parallel arrays Info, Left and Right and a pointer variable Root.
- Info[K]: Contains data at node N.
- Left[K]: Contains location of left child of N
- Right[K]: Contains location of right child of N
- Root: Contains location of Root
- Example:

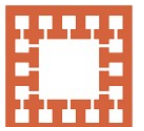


Root

	Info	Left	Right
1	C	0	10
2	D	0	0
3			
4	E	0	0
5	A	7	1
6			
7	B	2	4
8			
9			
10	F	0	0

Figure: Binary Tree T and its linked representation.

Sequential Representation of Binary Trees



CSE 2103

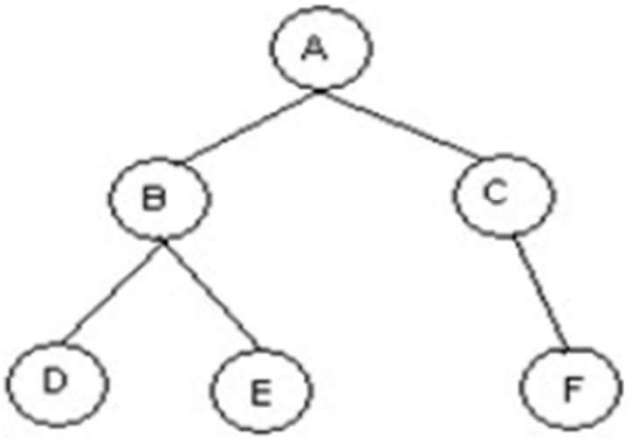
- Use only a single liner array Tree.

(a)Tree[1] represents the Root of T.

(b)If node N is in Tree[K], then its left child is in Tree[2K] and right child is in Tree[2K+1].

(c)Tree[1] = NULL indicates T is empty.

•Example:



Tree =

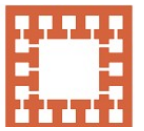
1	2	3	4	5	6	7	8	9	10
A	B	C	D	E		F			

Figure: Binary Tree T and its sequential representation.

Binary Tree Traversal Methods

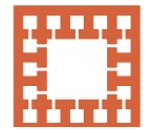
- In a traversal of a binary tree, each element of the binary tree is visited exactly once.
- During the visit of an element, all action (display, evaluate the operator, etc.) with respect to this element is taken.

Binary Tree Traversal Methods



- There are three standard ways of traversing a binary tree T with Root R:
  - **Preorder**
    - Process the Root R
    - Traverse the left subtree of R in preorder
    - Traverse the right subtree of R in preorder
  - **Inorder**
    - Traverse the left subtree of R in preorder
    - Process the root R
    - Traverse the right subtree of R in inorder
  - **Postorder**
    - Traverse the left subtree of R in postorder
    - Traverse the right subtree of R in postorder
    - Process the root R

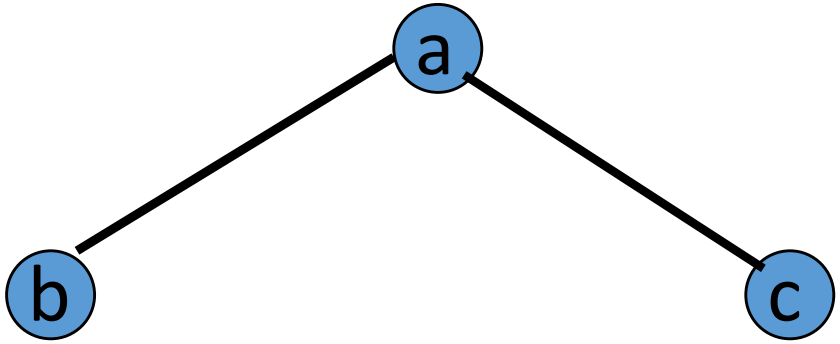
Binary Tree Traversal Methods



Preorder Traversal

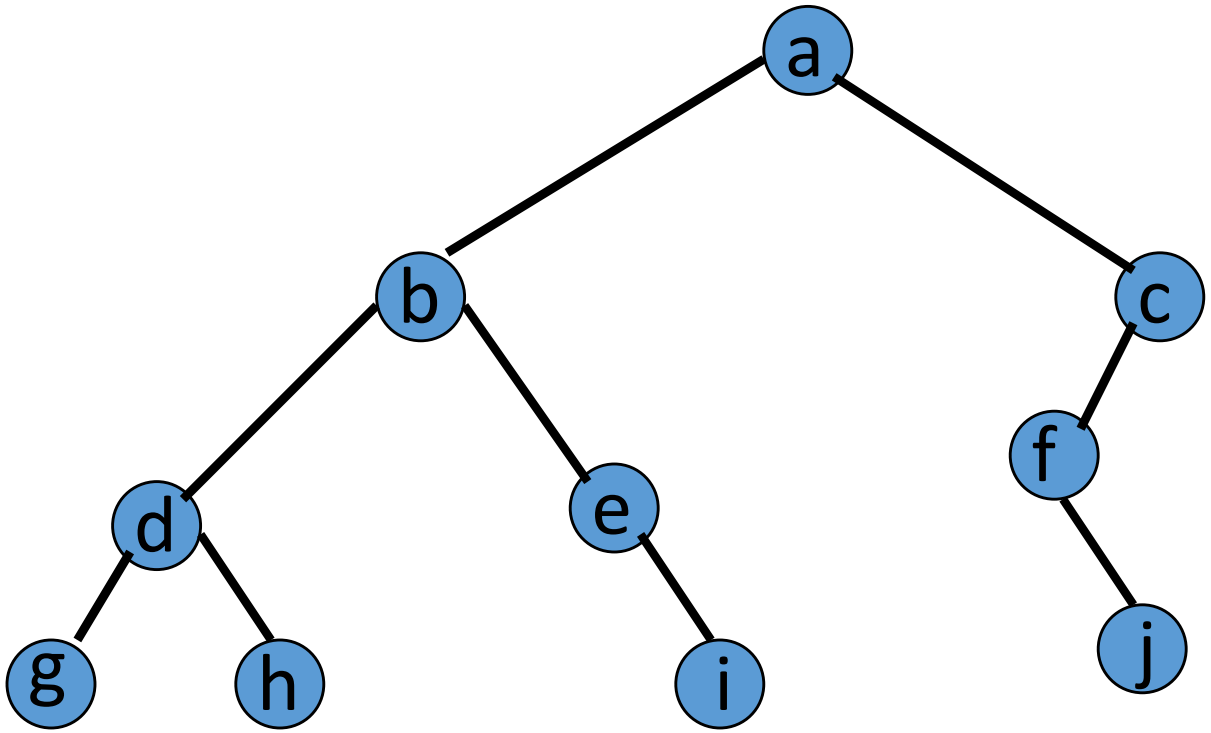
```
preOrder(treePointer ptr)
{
      (ptr != NULL)
   {
      visit(ptr);
      preOrder(ptr->leftChild);
      preOrder(ptr->rightChild);
   }
}
```

Preorder Example (Visit = print)



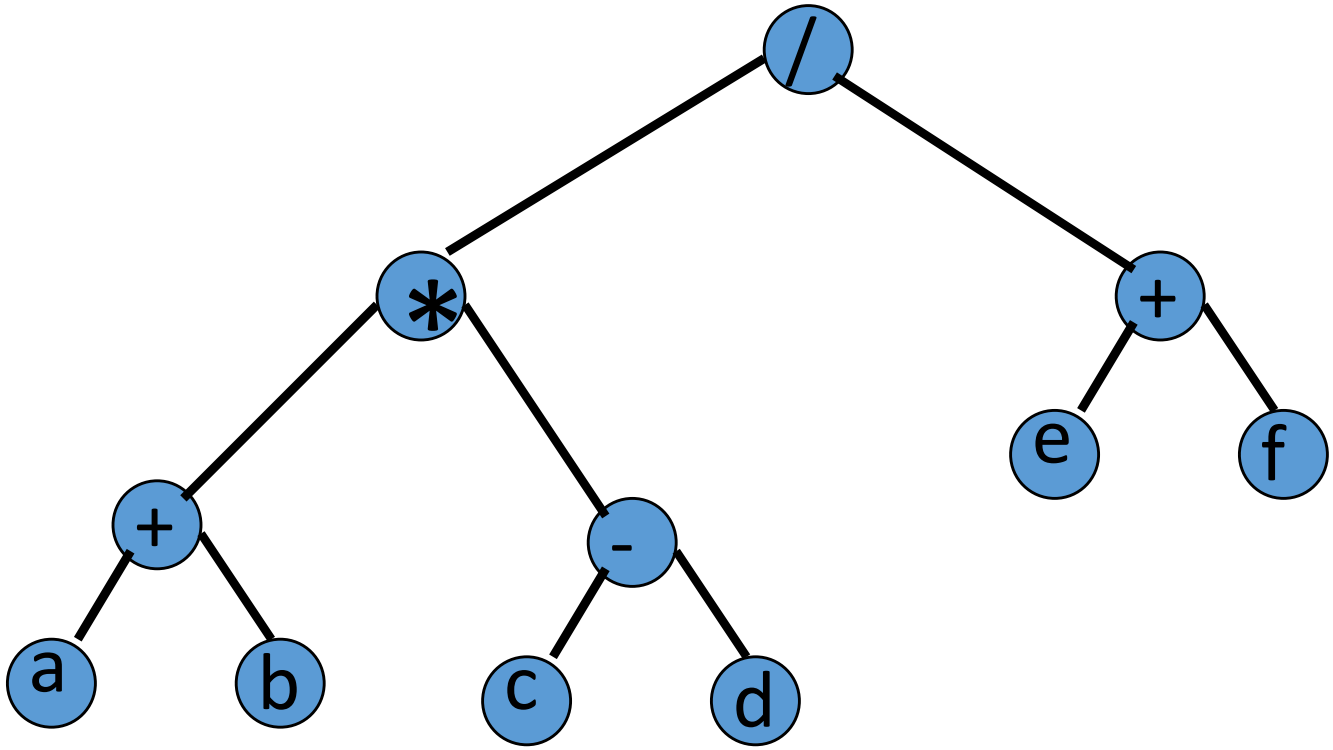
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

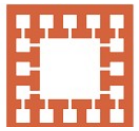
Preorder Of Expression Tree



/ *+ab- cd+e f

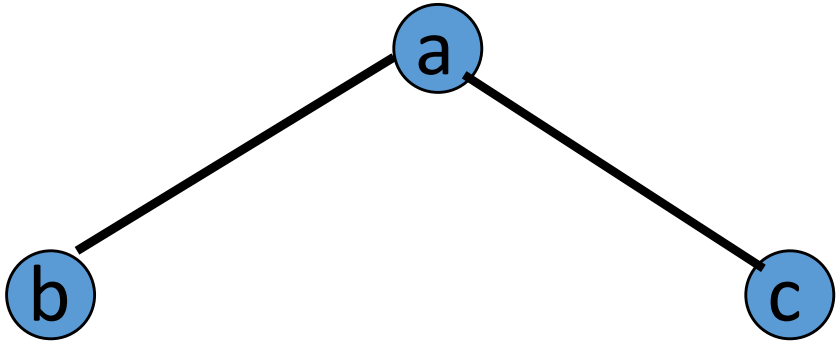
Gives prefix form of expression!

Inorder Traversal



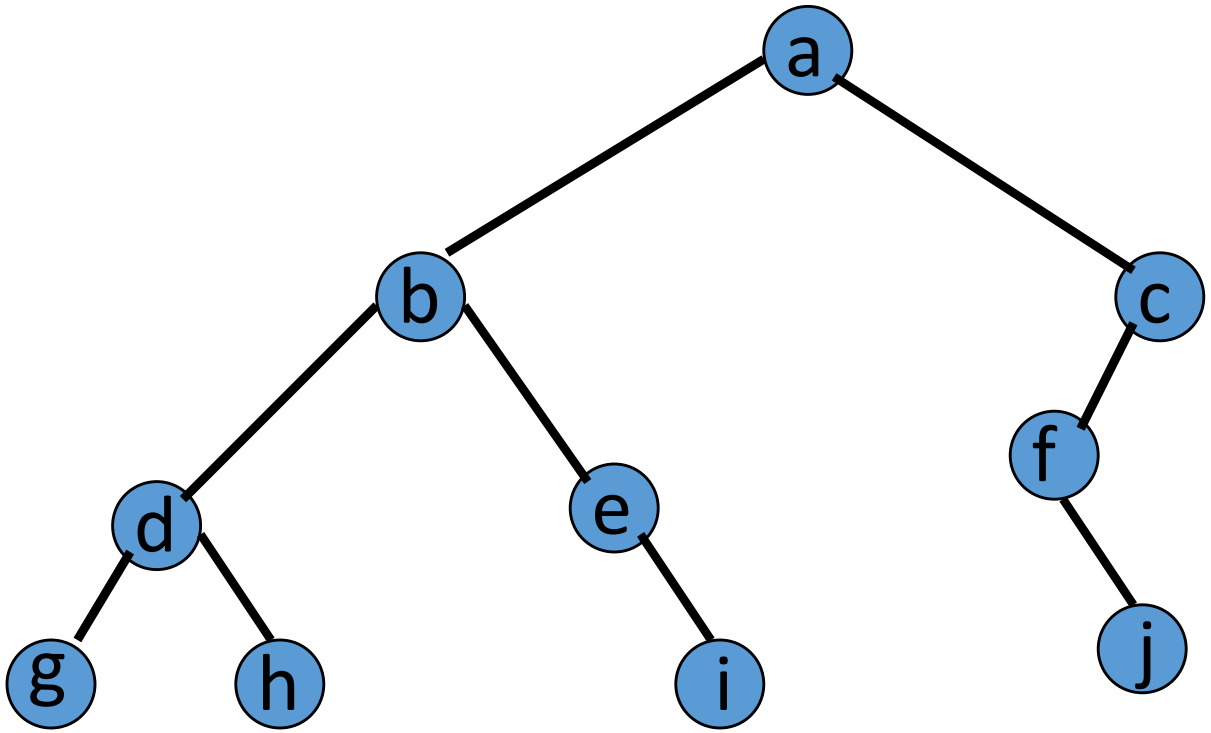

```
inOrder(treePointer ptr)
{
      (ptr != NULL)
   {
      inOrder(ptr->leftChild);
      visit(ptr);
      inOrder(ptr->rightChild);
   }
}
```

Inorder Example (Visit = print)



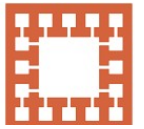
b a c

Inorder Example (Visit = print)



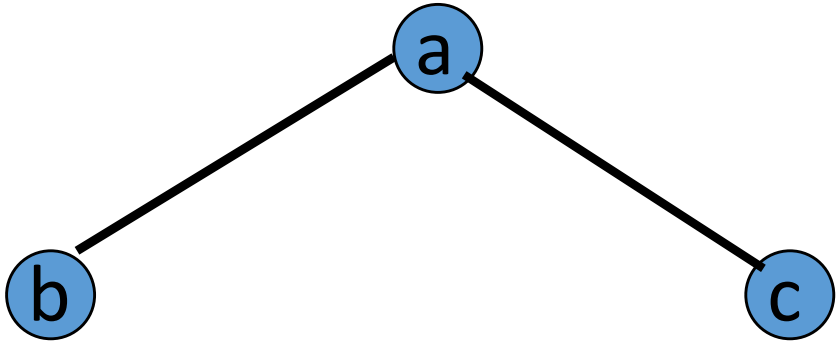
g d h b e i a f j c

Postorder Traversal



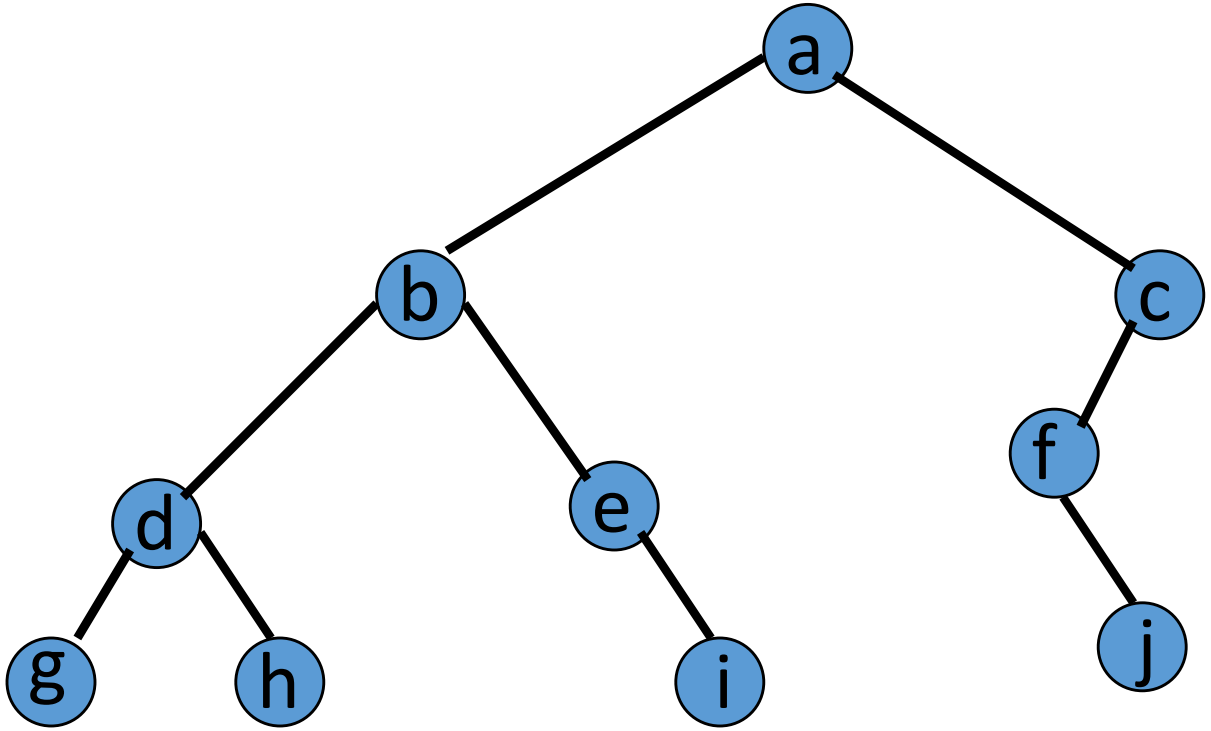

```
postOrder(treePointer ptr)
{
      (ptr != NULL)
   {
      postOrder(ptr->leftChild);
      postOrder(ptr->rightChild);
      visit(ptr);
   }
}
```

Postorder Example (Visit = print)



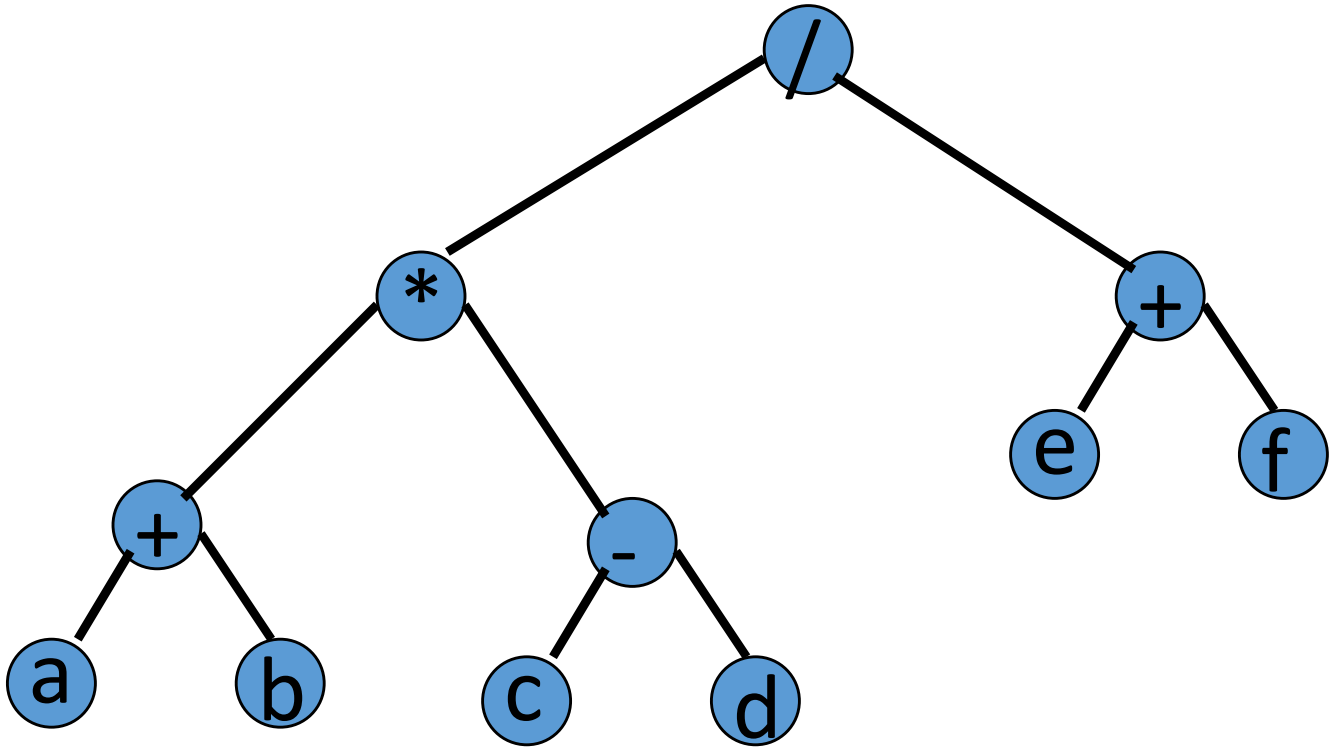
b c a

Postorder Example (Visit = print)



ghdi ebj f c a

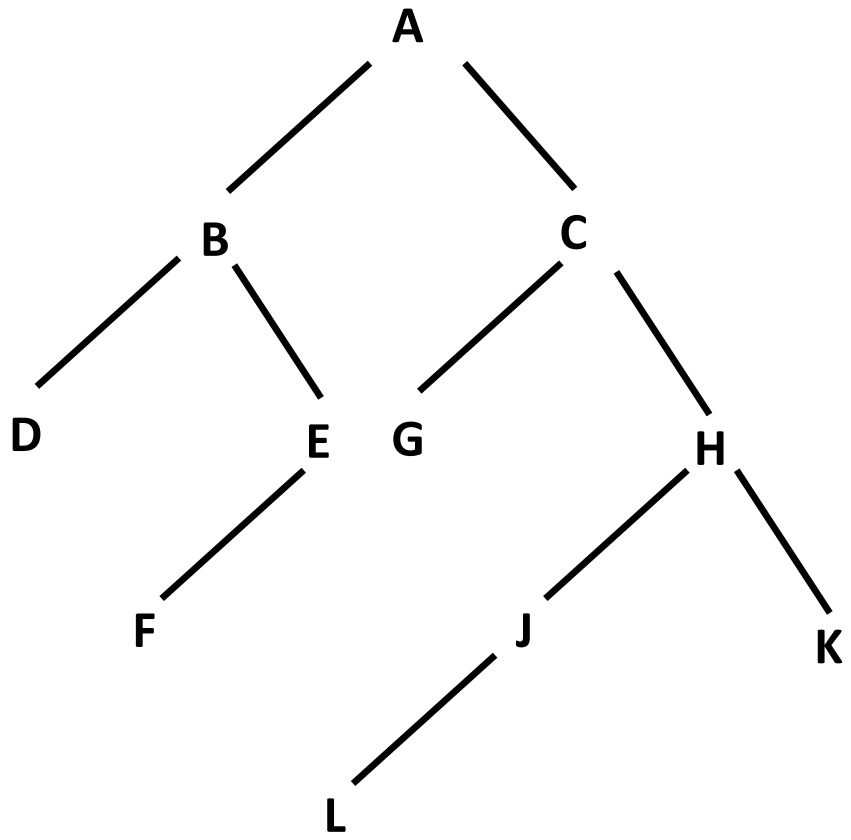
Postorder Of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression!

Traversal Applications



The Preorder traversal of T:

A B D E F C G H J L K

The Inorder traversal of T:

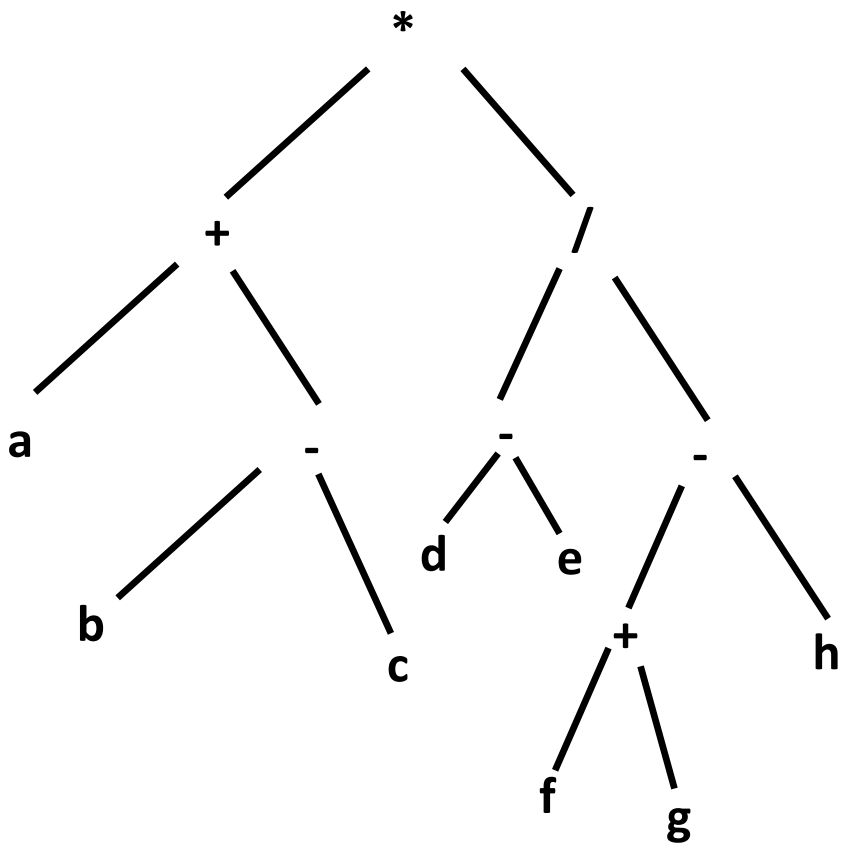
D B F E A G C L J H K

The Postorder traversal of T:

D F E B G L J K H C A

Traversal Applications

[a+(b-c)]*[(d-e)/(f+g-h)]



The Preorder traversal of T:

*** + a – b c / -d e - + f g h**

The Postorder traversal of T:

a b c - + d e – f g + h - / *