Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE

INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

YEAR	SEM
2	3

CS8391

DATA STRUCTURES
(Common to CSE & IT)

UNIT No. 3

NON-LINEAR STRUCTURES - TREES

- BINARY TREE ADT
- EXPRESSION TREES

Version: 1.XX

Non-Linear Data Structures

The data structure where data items are not organized sequentially is called non linear data structure.

- A data item in a nonlinear data structure could be attached to several other data elements to reflect a special relationship among them and all the data items cannot be traversed in a single run.

Data structures like trees and graphs are some examples of widely used nonlinear data structures.

Binary Tree

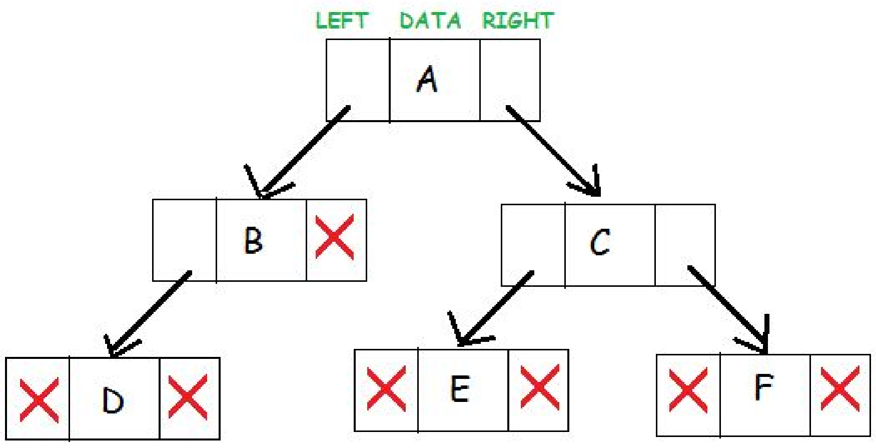
A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.

Each node contains three components:

- Pointer to left subtree
- Pointer to right subtree
- Data element

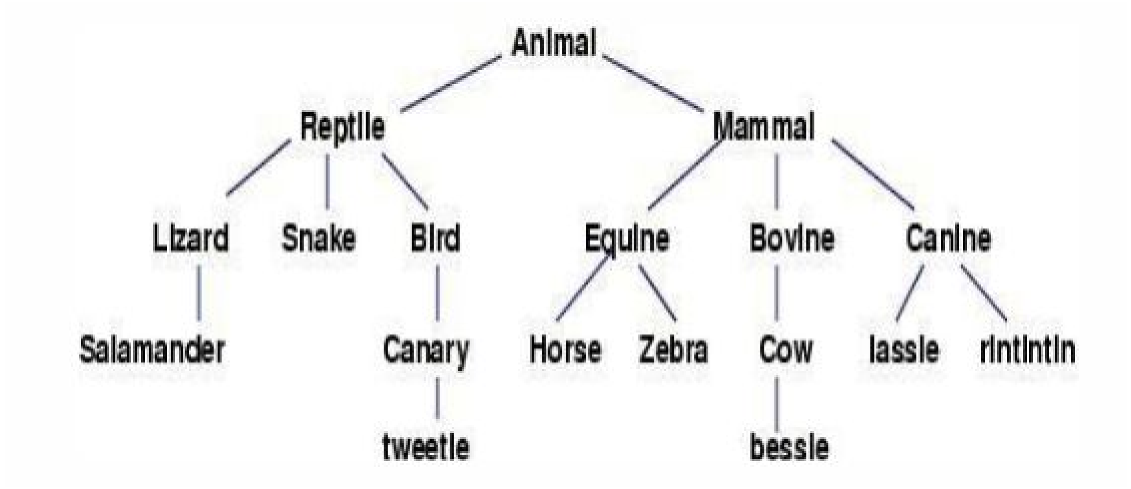
The topmost node in the tree is called the root. An empty tree is represented by a NULL pointer.

A representation of binary tree is shown:



Example

Tree of Species

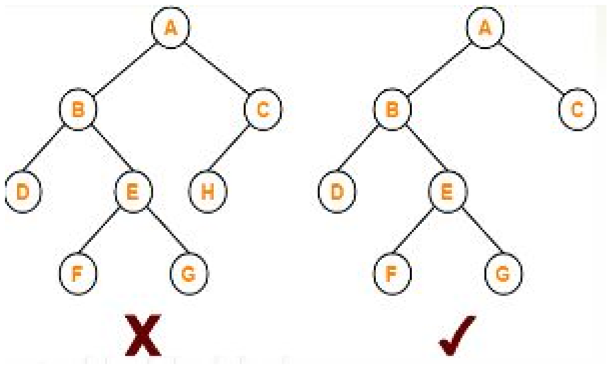


Types

Full/Strictly Binary Tree

A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree.

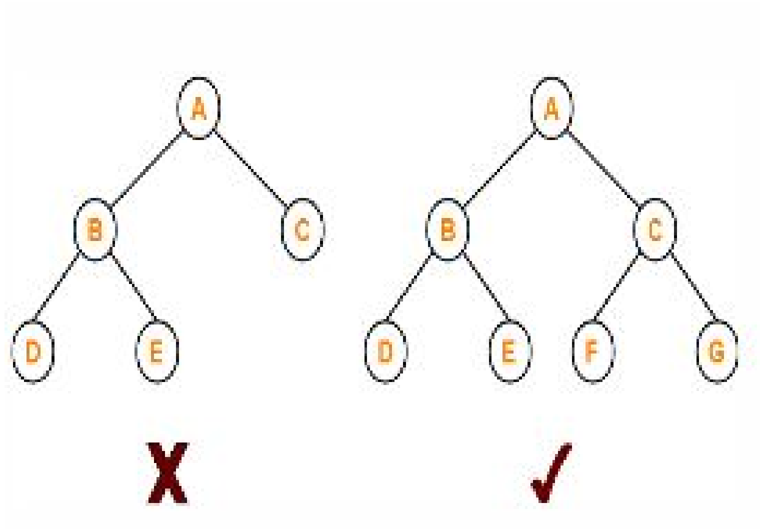
Strictly binary tree is also called a **Full Binary Tree** or **Proper Binary Tree** or **2-Tree.**



Perfect Binary Tree

A Perfect 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.

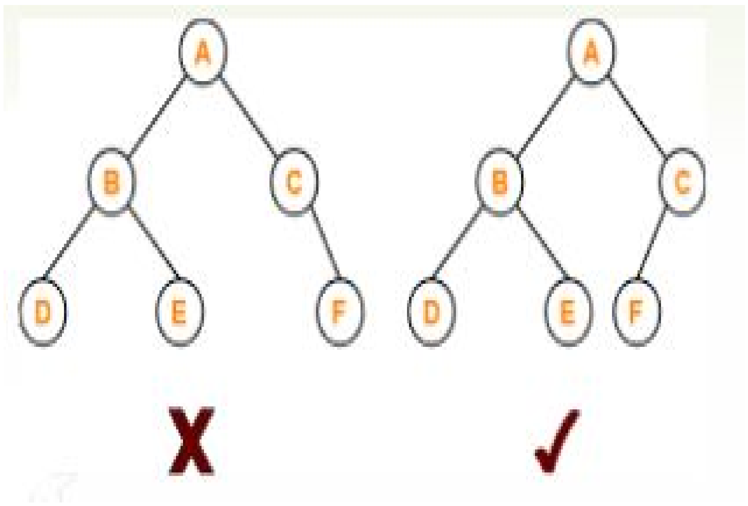


Complete Binary Tree

A **complete binary tree** is a binary tree that satisfies two properties

- All the levels are completely filled except possibly the last level
- The last level must be strictly filled from left to right

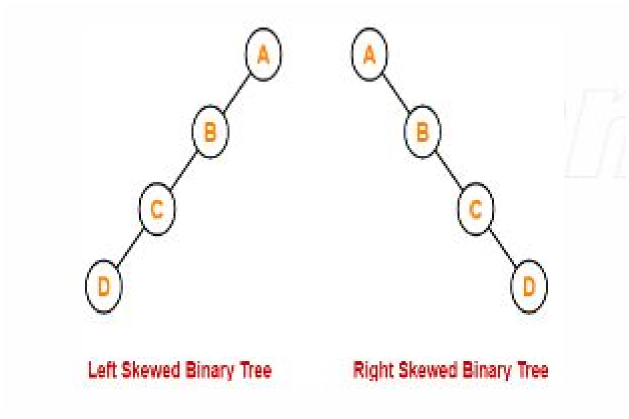
Complete binary tree is also called as **Perfect Binary Tree**



Skewed Binary Tree

A Skewed binary tree is a binary tree that satisfies the following 2 properties-

- All the nodes except one node has one and only child.
- The remaining nodes have no child.

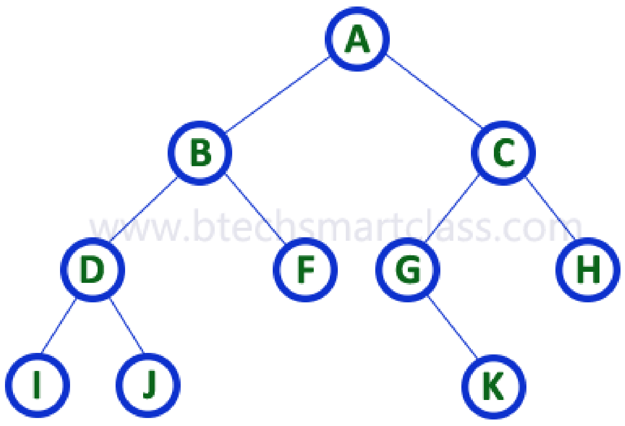


Binary Tree Representation

A binary tree data structure is represented using two methods. Those methods are as follows...

1. **Array Representation**
2. **Linked List Representation**

Consider the following binary tree...

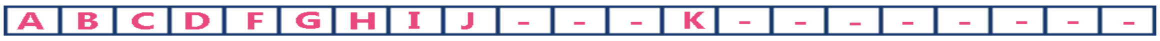


Array Representation of Binary Tree

In an array representation of a binary tree, we use a one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...

To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of **2n + 1**.

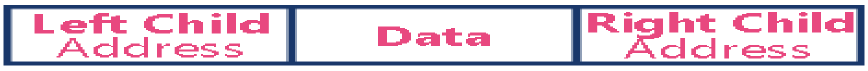


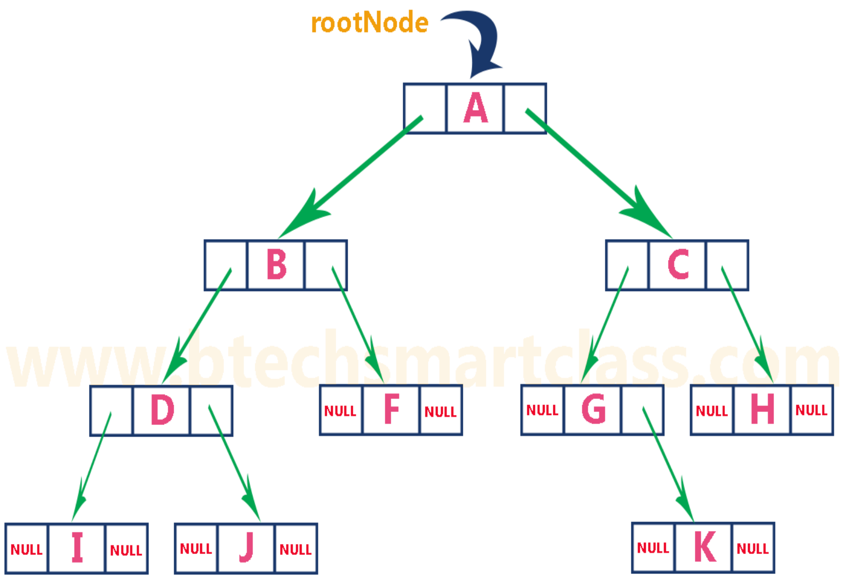
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child addresses, second for storing actual data and third for storing right child addresses.

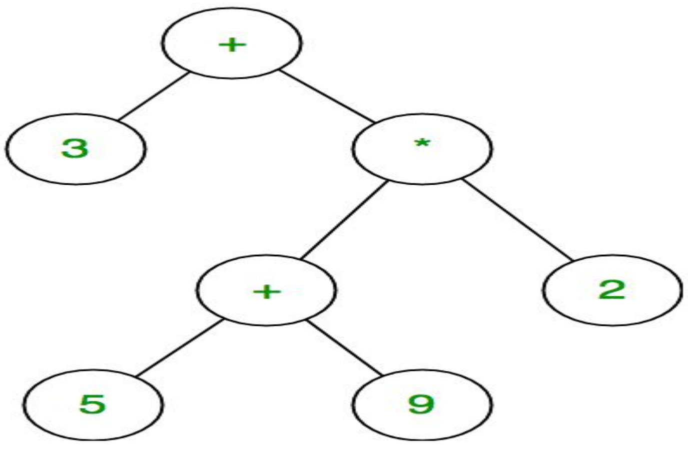
In this linked list representation, a node has the following structure...





Expression Trees

- Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be:
- Inorder traversal of expression tree produces infix version of given postfix expression
- Expression Tree is a special kind of binary tree with the following properties:
  1) Each leaf is an operand.
  2) The root and internal nodes are operators.
  3) Subtrees are subexpressions with the root being an operator.



Construction of Expression Tree

- We consider that a postfix expression is given as an input for constructing an expression tree. Following are the step to construct an expression tree:
  1. Read one symbol at a time from the postfix expression.
  2. Check if the symbol is an operand or operator.
  3. If the symbol is an operand, create a one node tree and pushed a pointer onto a stack
  4. If the symbol is an operator, pop two pointer from the stack namely T1 & T2 and form a new tree with root as the operator, T1 & T2 as a left and right child
  5. A pointer to this new tree is pushed onto the stack

Postfix to Expression Tree Rules

Terms: Append x to y means add child x to node y.

Rule1: Operators can have children but operands can't.

Rule2: Nodes can only have 2 children.

Rule3: When appending nodes, always append to right first. If right is occupied, then append to left.

Postfix to Expr Tree Algorithm

1.Get the last symbol (rightmost) of postfix notation, create a node for it and designate the new node as the root.

2.Set the root node as the current node.

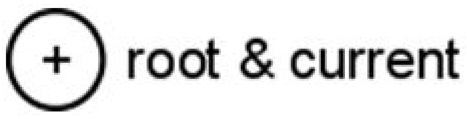
3.For each element from right to left (excluding the last):

- Create a node for it.
- If the current node cannot have more children, search for the first parent/grandparent that can have more children and set it as the current node.
- Append the new node to the current node.
- Set new node as the current node.

Postfix to Exp Tree (Step by Step)

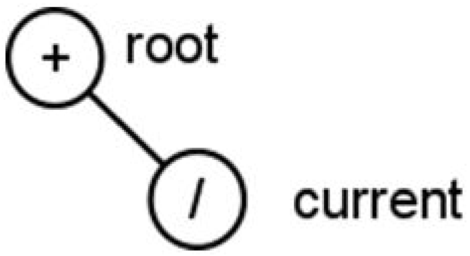
Postfix: 2, 6, *, 3, 8, /, +

Step 1 & 2



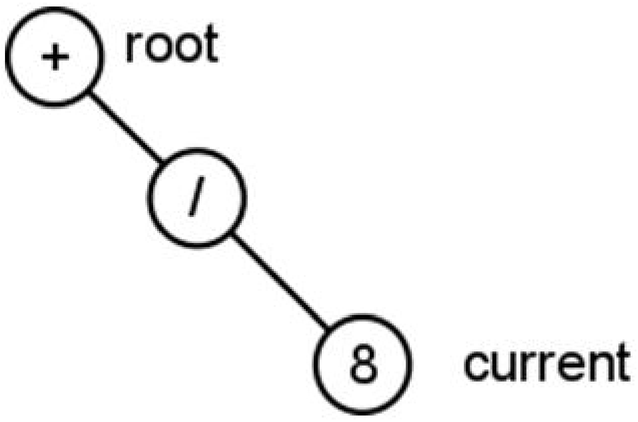
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1, 3.3 & 3.4



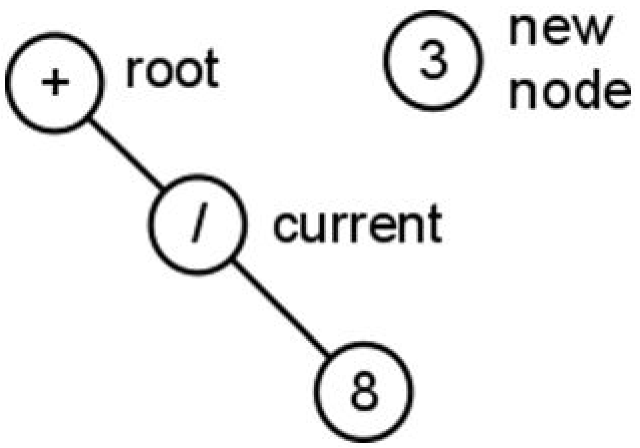
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1, 3.3 & 3.4



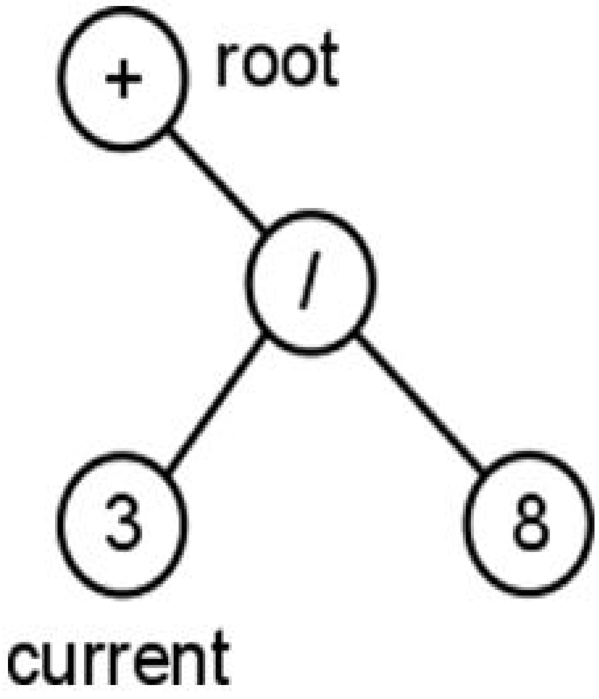
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1 & 3.2



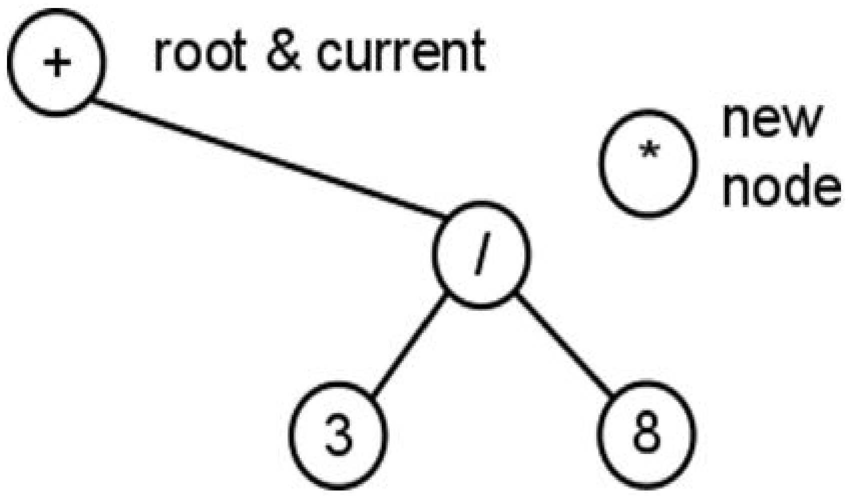
Postfix: 2, 6, *, 3, 8, /, +

Step 3.3 & 3.4



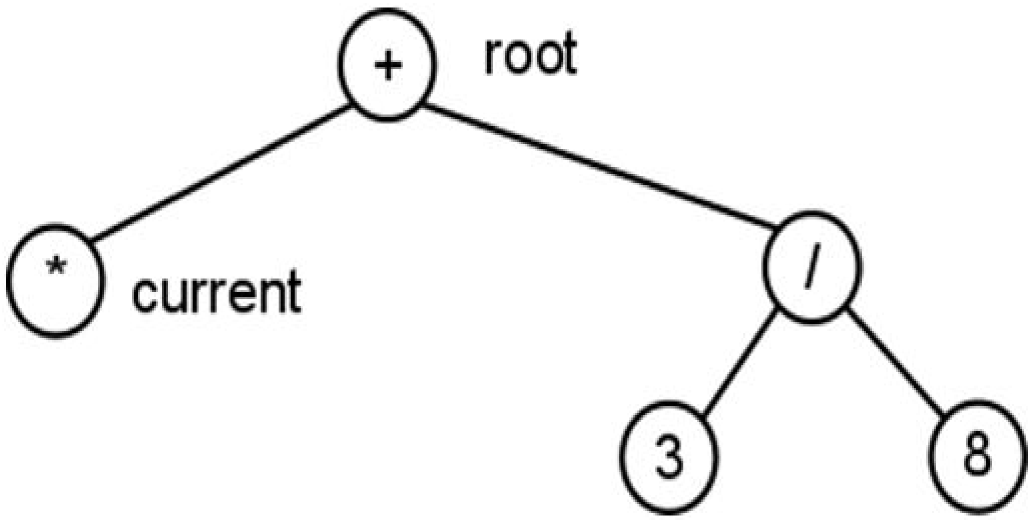
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1 & 3.2



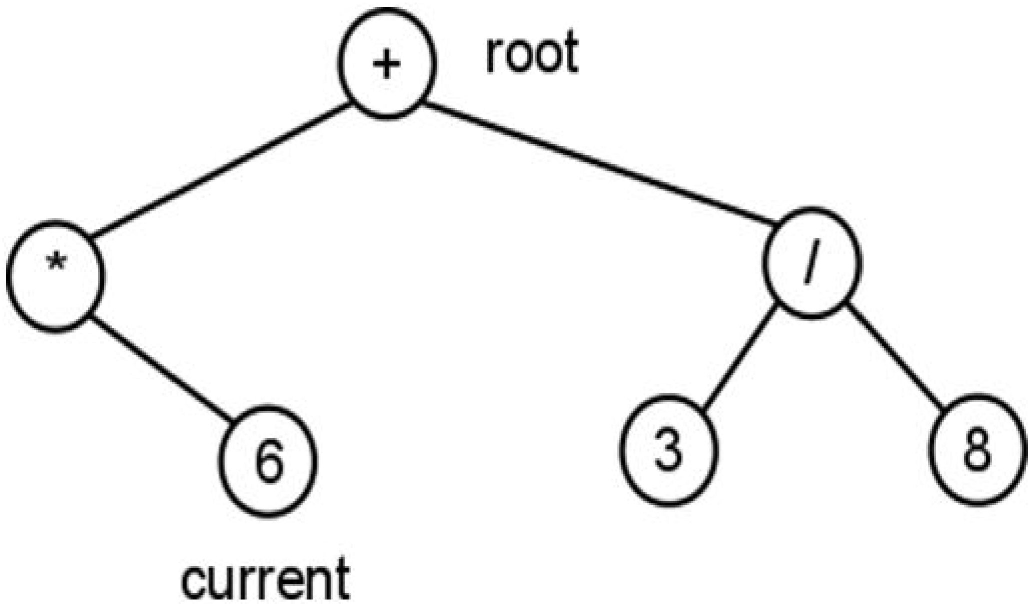
Postfix: 2, 6, *, 3, 8, /, +

Step 3.3 & 3.4



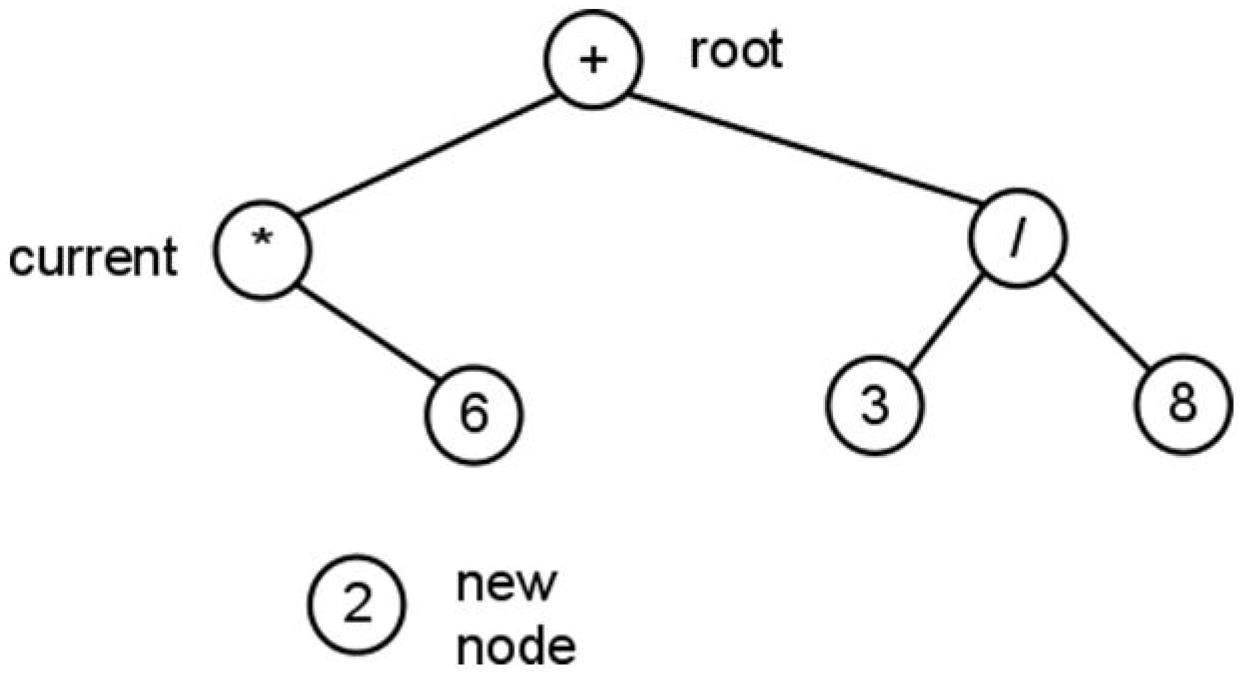
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1, 3.3 & 3.4



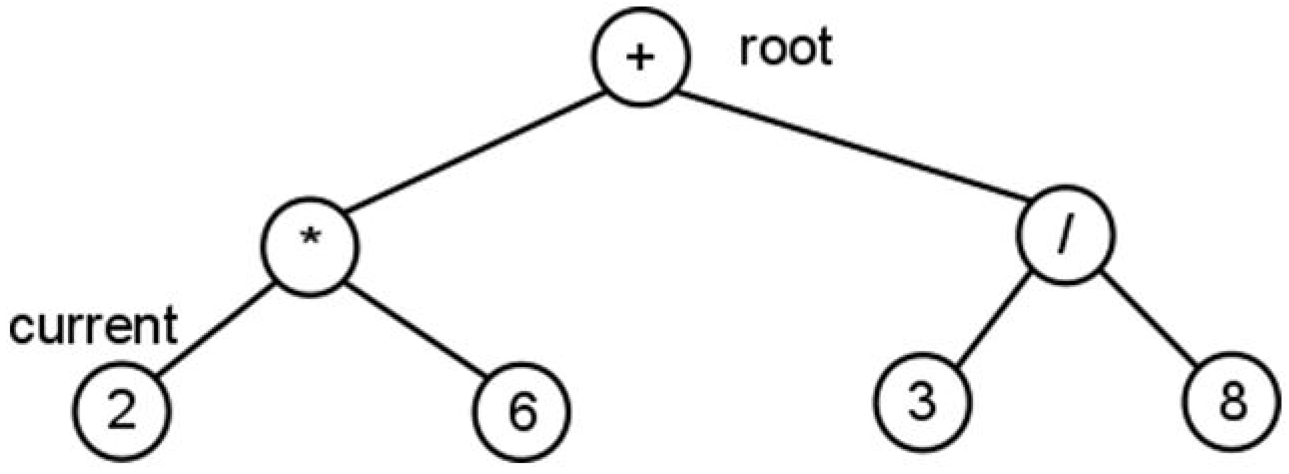
Postfix: 2, 6, *, 3, 8, /, +

Step 3.1 & 3.2



Postfix: 2, 6, *, 3, 8, /, +

Step 3.3 & 3.4



Prefix from Expression Tree Algorithm

Rule1: Start at the root node.

Rule2: When node is visited for the first time, output value of node.

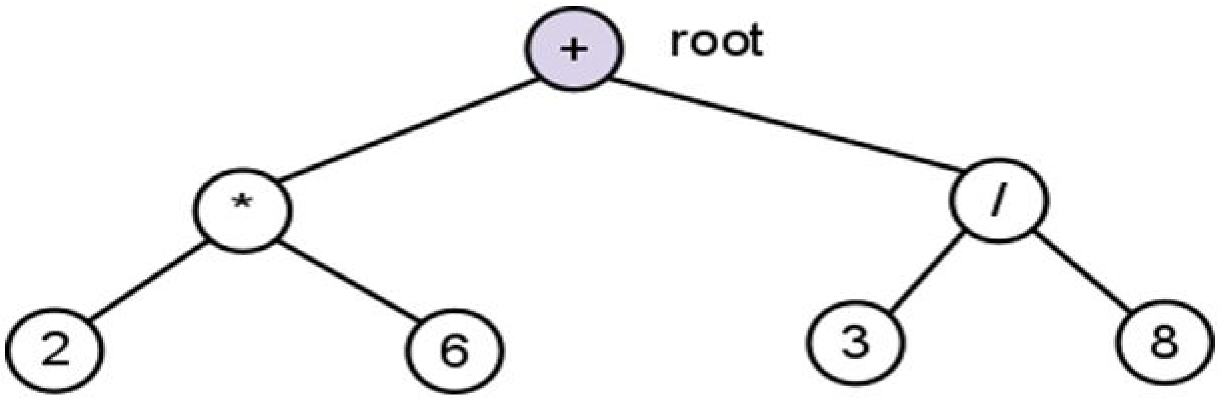
Rule3: Go from left to right when visiting children.

Rule4: If left child has children, visit their children first before going to right child.

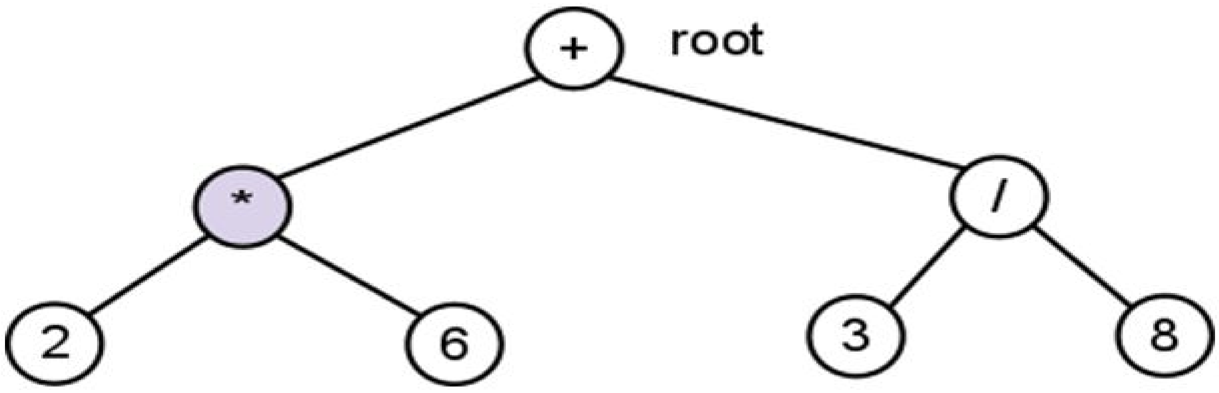
Rule5: Prefix notation is complete when every node is visited.

Prefix from Exp Tree (Step by Step)

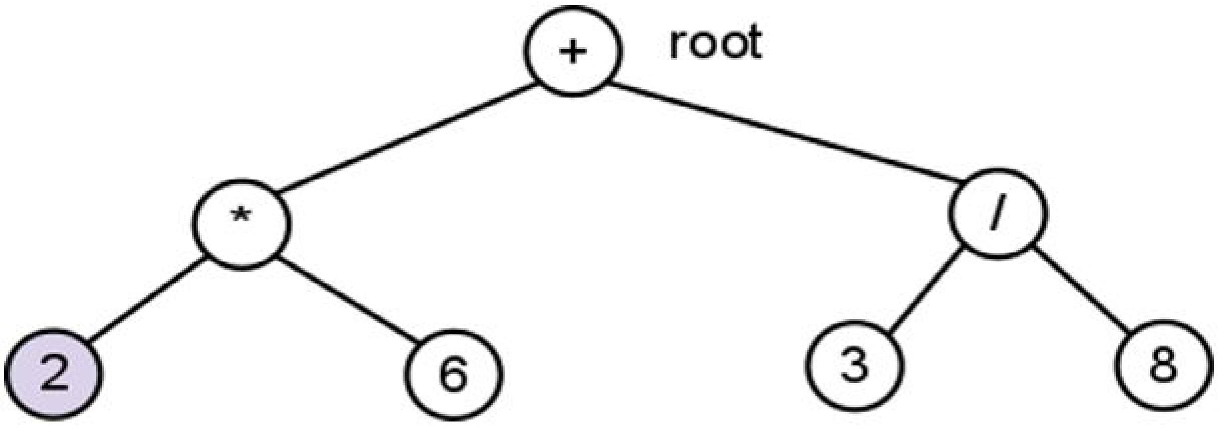
Output: +



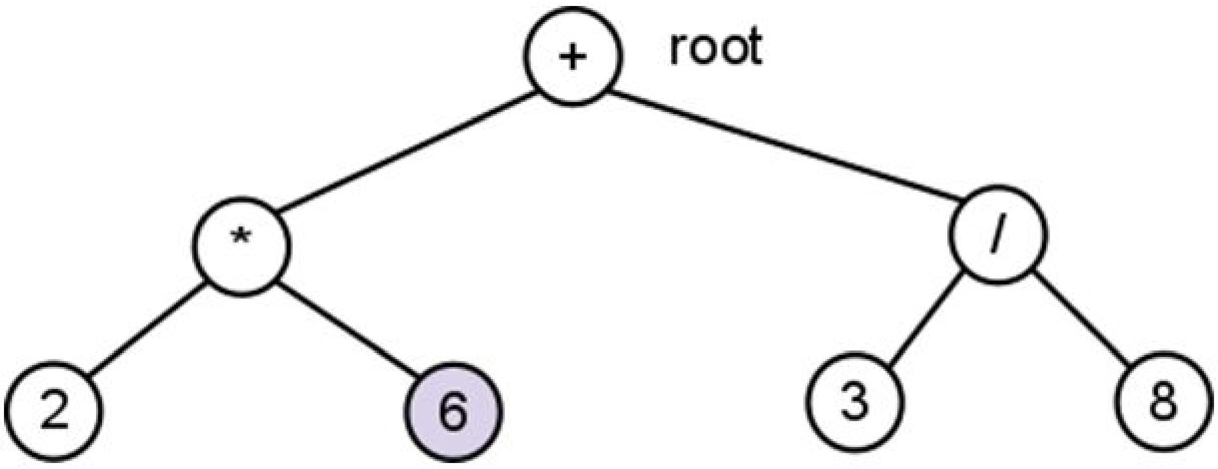
Output: +, *



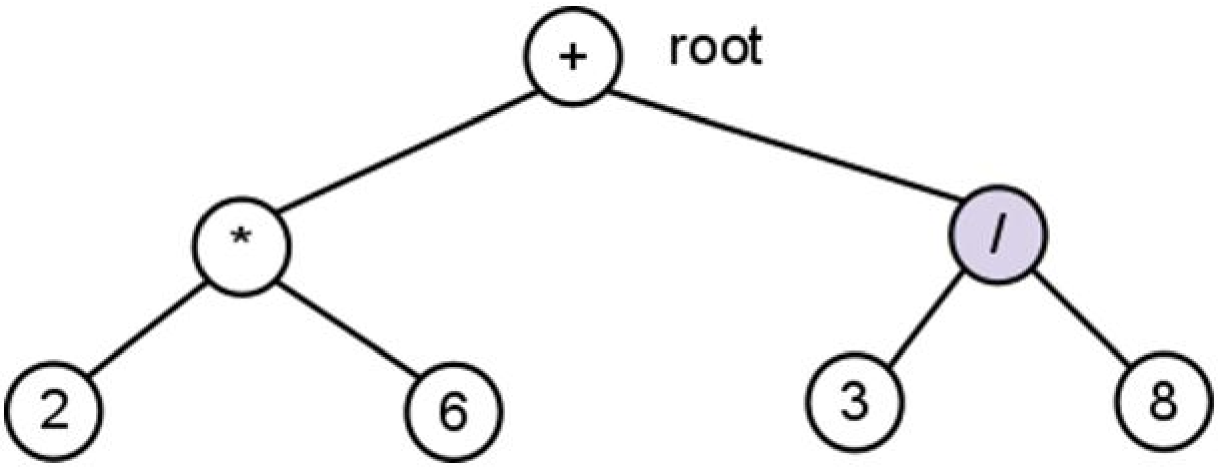
Output: +, *, 2



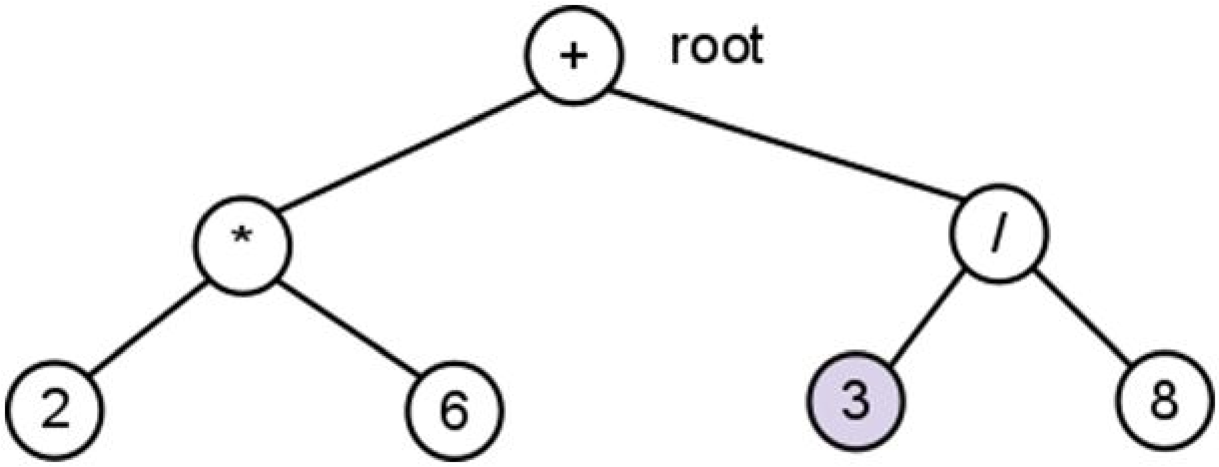
Output: +, *, 2, 6



Output: +, *, 2, 6, /



Output: +, *, 2, 6, /, 3



Output: +, *, 2, 6, /, 3, 8