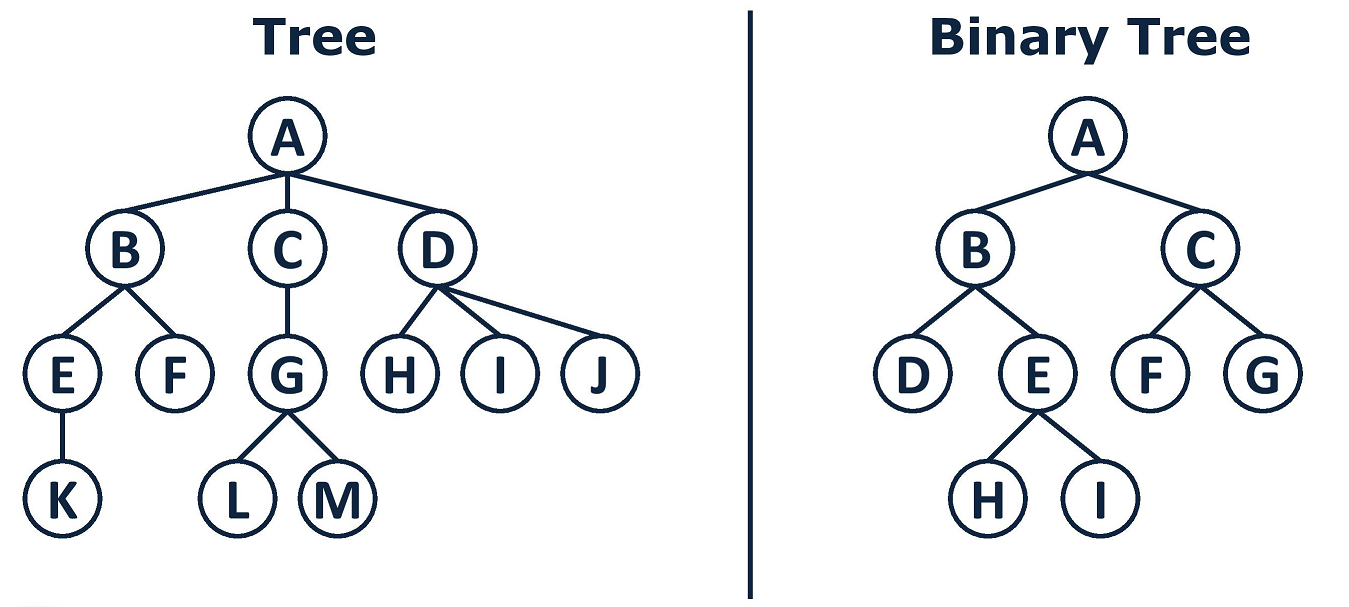
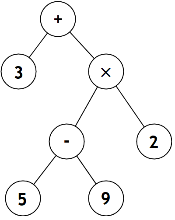
It is difficult to overstate the importance of *Tree* structures in engineering. The two examples appearing below support the following terminology and graphical criteria,

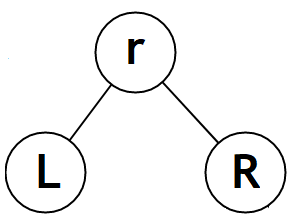
1. Data appear in circles or squares called *nodes*
2. The *root* node is placed at the top
3. The *parent-child* relationship is depicted by a line segment
4. Each *child* has only one parent
5. A *leaf* node is one without a child
6. An *inner* node has at least one child
7. A *binary tree* (below, right) is a tree that has, at most, two children
8. Each *inner* node can be considered as the root of its own *subtree*.



This introduction focuses on the tree’s implementation as a *binary expression tree* (BET) in support of our RPN Calculator Project. A BET is a graphical representation of an arithmetic expression in which the *inner* nodes hold operators and the *leaf* nodes hold operands.

The BET for appears to the right. This BET has four leaf nodes and three inner nodes.

Notice immediately that a BET does not include brackets as the order of operator precedence is accounted for in the way one *traverses* (evaluates) the tree.



To prepare for a discussion of traversal options, consider the generic BET structure to the right. This BET has a root node, **r**, a left child, **L**, and a right child, **R**. There are three conventional ways to traverse a BET. These are,

* **Pre**order: the root is visited first, followed by the left child and right child, **rLR**
* **In**order: the left child is visited first, followed by the root and right child, **LrR**
* **Post**Order: the left child is visited first, followed by the right child and root, **LRr**

**Exercise**

For each row below, complete the missing three entries.

|  |  |  |  |
| --- | --- | --- | --- |
| **Preorder** | **Inorder** | **Postorder** | **Binary (Expression) Tree** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |