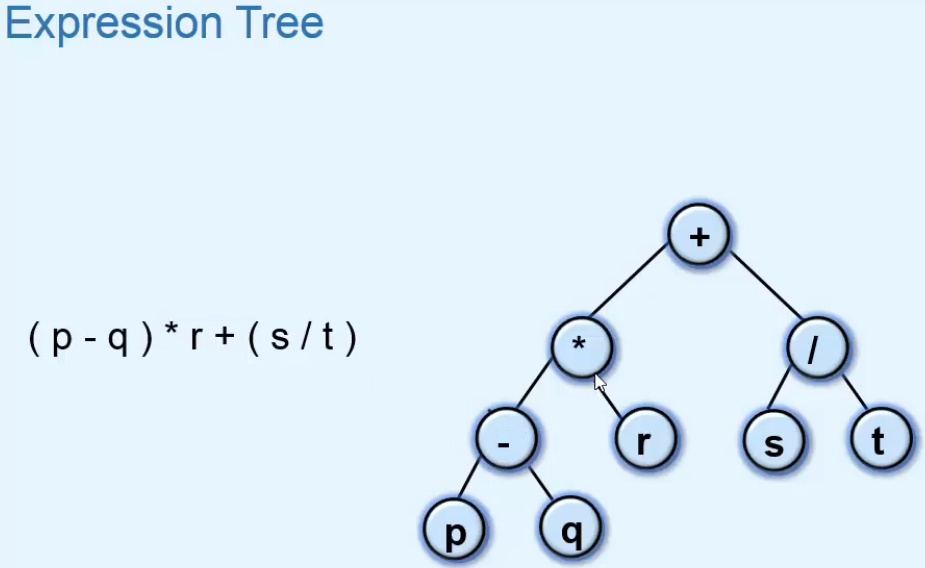
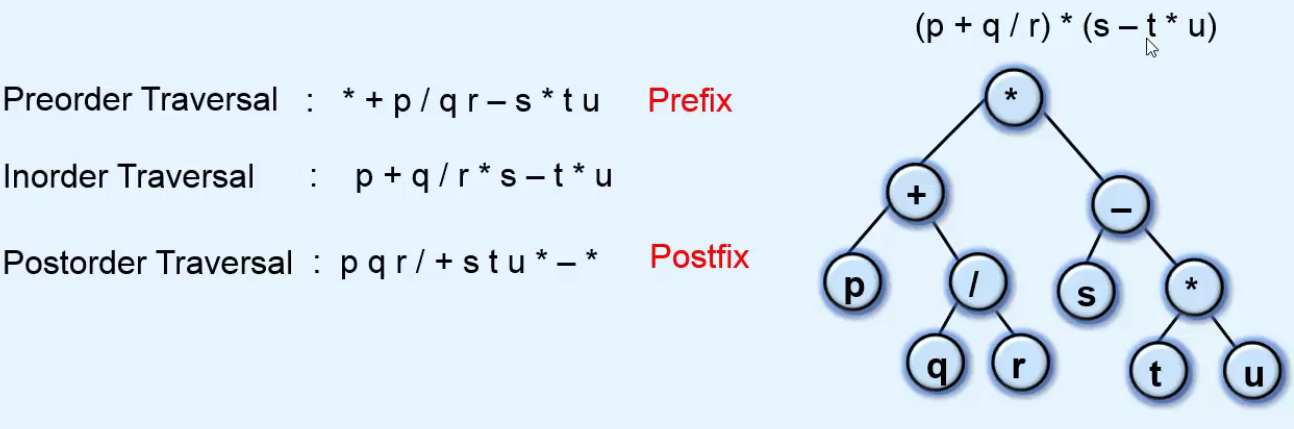
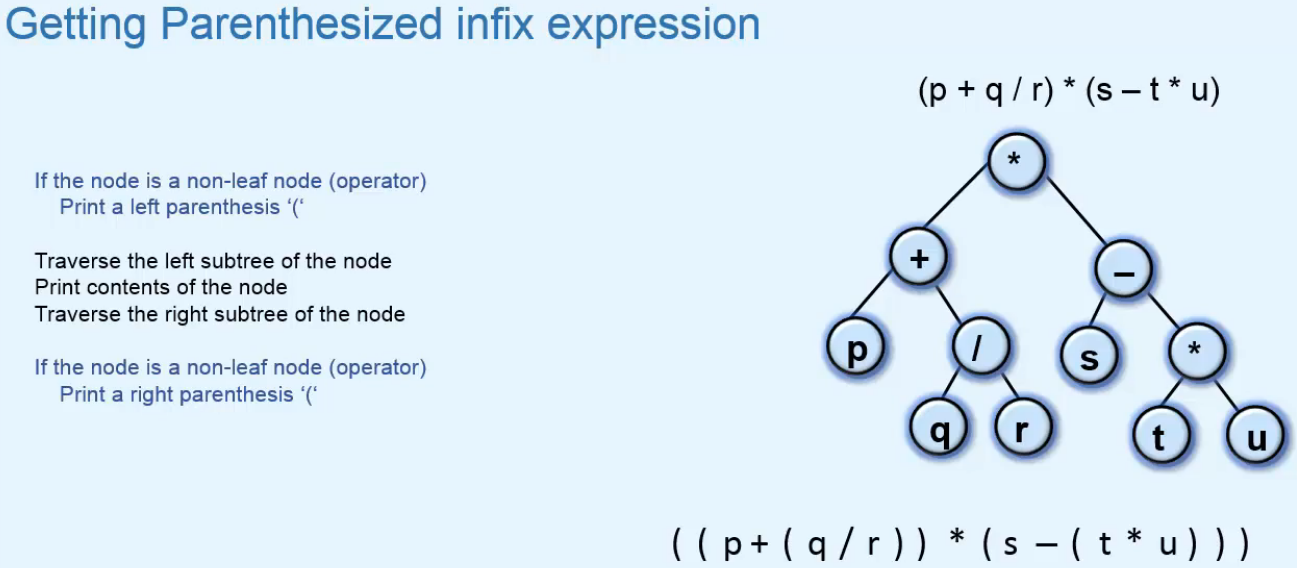
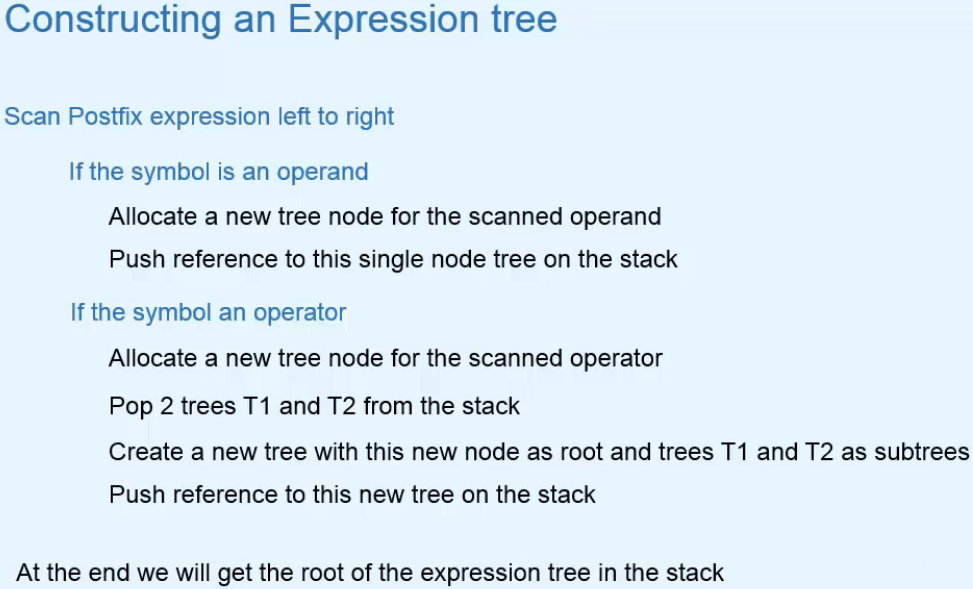
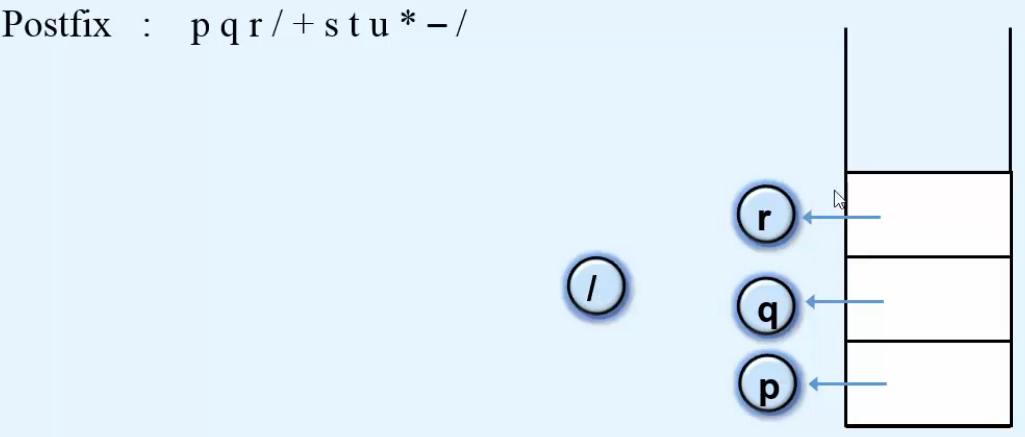
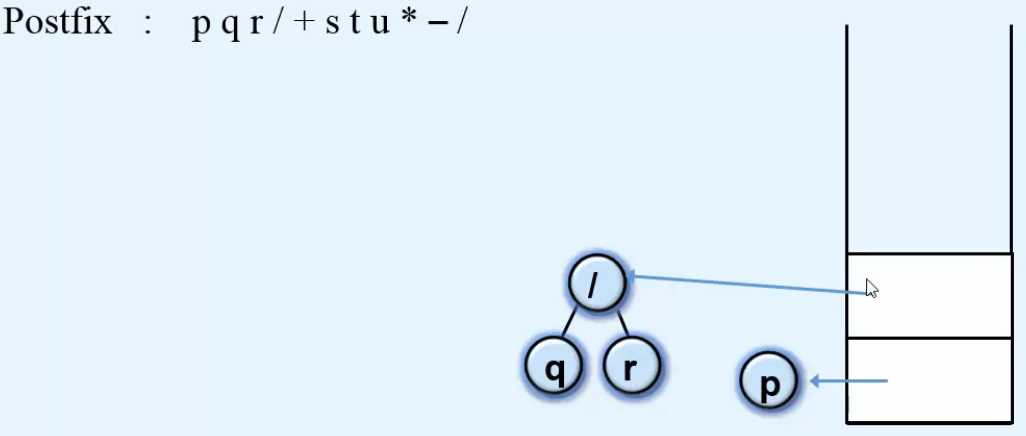
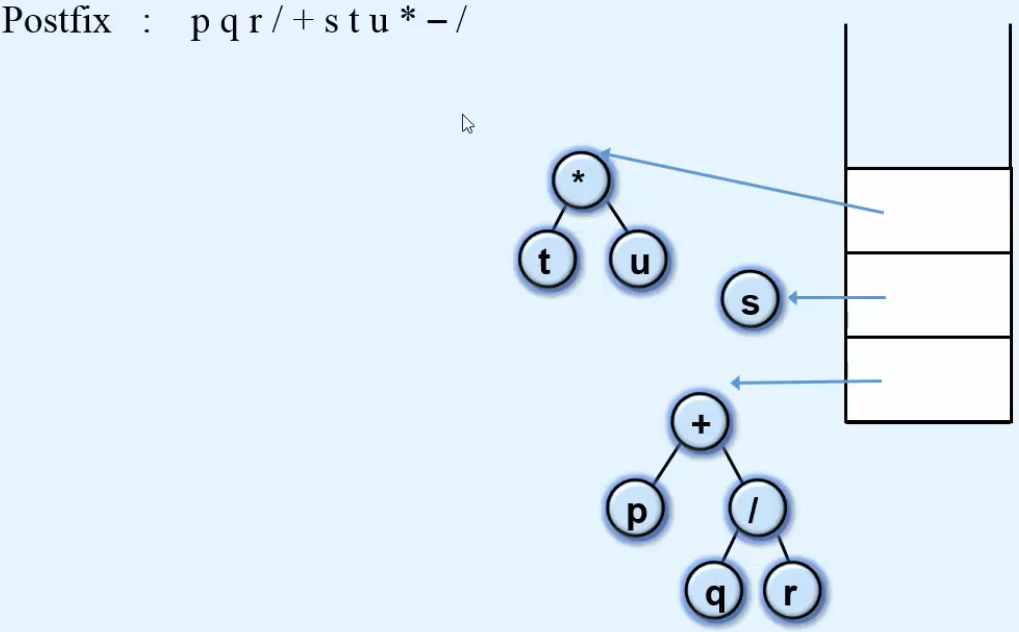
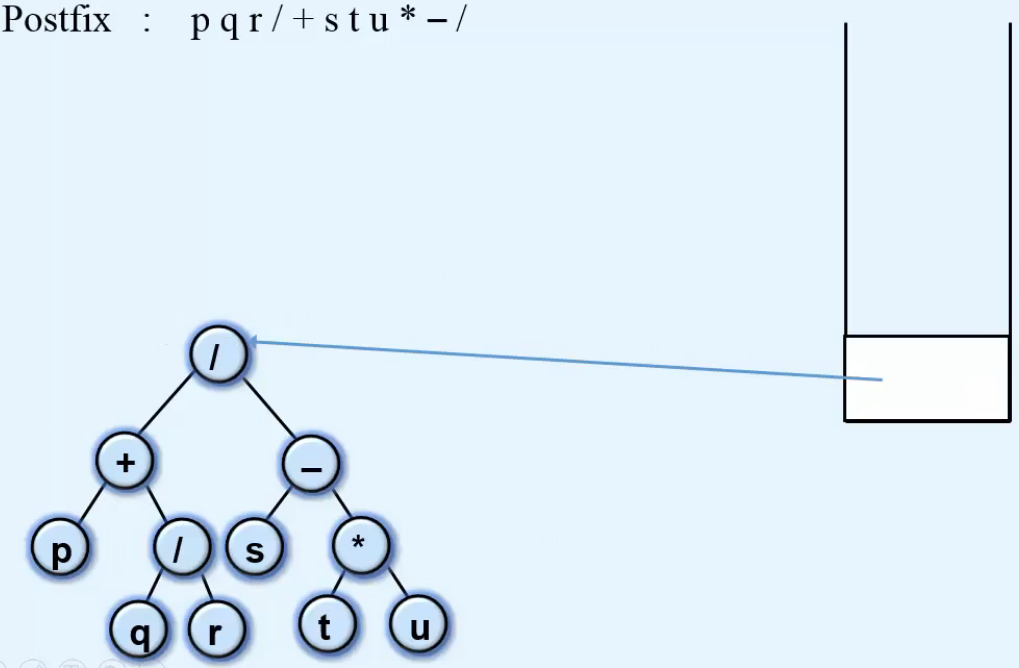
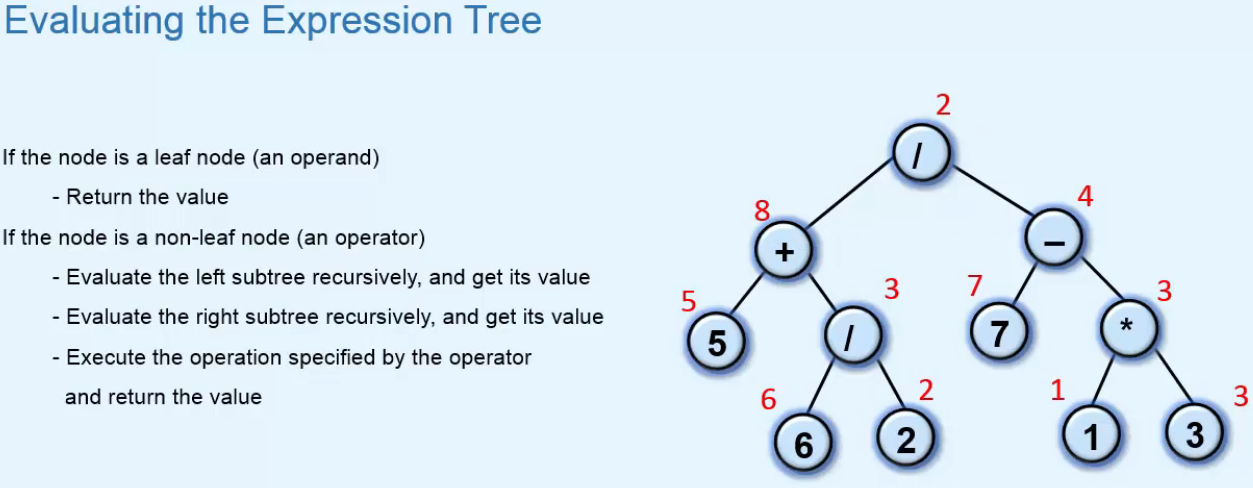
**Introduction**  
**Expression Tree** => **Binary Tree that represents an algebric or a boolean expression**.  
  
=> **Non-leaf nodes in this tree correspond to the operators**.  
=> **Leaf nodes correspond to the left/right operands**.  
\* **Arithmetic and boolean operators unary or binary so the Expression Trees are generally Binary Trees**.  
**Unuary operators** => Negation or Boolean NOT operator, the operator node will have only 1 child.  
=> **The operators at the lower level are evaluated before the operators at the higher level**.  
\* In our case:  
-  
\* /  
+  
  
\* **The parentheses are not a part of the tree but the tree retains the purpose of parentheses**.  
**Traversal** => Expression Tree is a Binary Tree and we can traverse a Binary Tree in 3 standard ways:  
  
\* We can get the fully parenthesized version by doing some extra work.  
  
**Constructing and Evaluating an Expression Tree**  
\* Expression Tree can be easily constructed from Postfix Expression or Prefix Expression.  
  
   
   
\* **When popping 2 items, the first one popped will be the right subtree**.  


**Implementation of Expression Tree**  
\* Our implementation works only for single digit operands, we’re keeping it simple so that we can focus on the algorithm.  
