# Lecture 11.3

**Binary Tree and Reverse Polish Notation**

## Infix Notation

- An arithmetic calculator evaluates an expression such as 5+7 * 3 to determine that it is equal to 26.
- This kind of expression is know as ***infix*** because the *operator* (+) is written between two *operands* (5,7)

## Infix Notation

- This notation is not very clear every time:
- Example:

  What is the result of

  3 – 4 – 5 * 3 = ?

  evaluating from left to right : -18

  evaluating according to precedence rules: -16

## Reverse Polish Notation

- We can use a different notation to make an expression more clear without using parenthesis.
- **Reverse Polish Notation** (RPN) is also known as **postfix** notation

## Reverse Polish Notation

- Instead of having an operator **between** two operands, you will have an operator **after** two operands
- Instead of: 3 – 4 – 5 * 3
- You have: 34 –53*-

## Evaluation rules for RPN

- Evaluate expressions from left to right
- At each occurrence of an operator, apply it to the two operands to the immediate left, and replace the sequence of two operands and one operator by the resulting value.

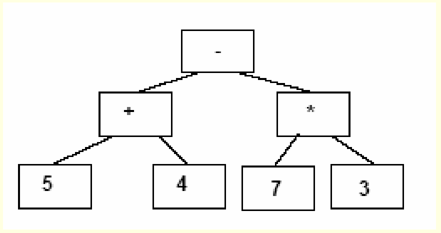
The expression : 34 –53*-
Evaluates to (3-4) (5*3)-
-1 15-
(-1 -15) = -16

## Implementation with a binary Tree

- For example:

  If your given expression = 54 + 73 * -

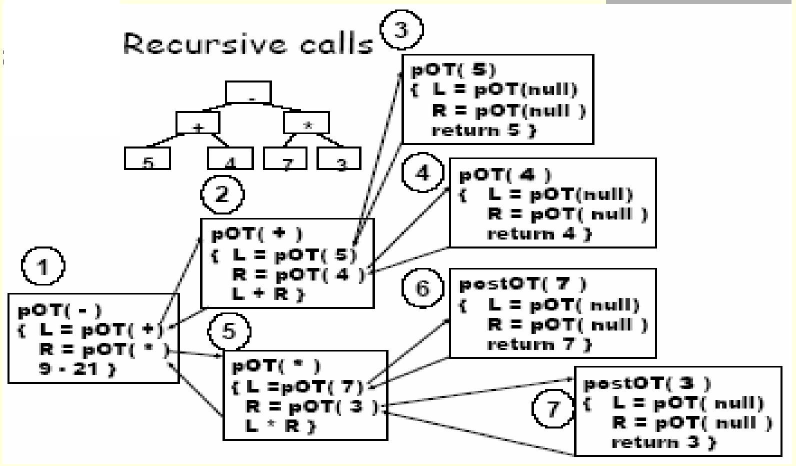
  Your can build a tree to represent the expression



- Write a function to evaluate this expression
- Hint: Which traversal scheme would help you?

# PostOrderEvaluate

```
double postOrderEvaluate ( BinaryTree <dataType> * bt )
{
     double left, right;
     if ( bt != NULL )
     {
              //traverse left child
              left = postOrderEvaluate ( bt->left ( ) );
              //traverse right child
              right = postOrderEvaluate ( bt->right( ) );
              //visit tree
              if ( strcmp(bt -> getData(), "+") == 0 )
              {             cout << endl << left << "+" << right;
                            return (left + right); }
              else if ( strcmp(bt -> getData(), "-") == 0)
              {             cout << endl << left << "-" << right;
                            return (left - right); }
              else if ( strcmp(bt -> getData(),"*" ) == 0)
              {             cout << endl << left << "*" << right;
                            return (left * right); }
              else
                            return atof(bt -> getData());
     }
   return 0;
}
```



```
pOT( - )
{ L = pOT( + )
  R = pOT( * )
  9 - 21 }

pOT( + )
{ L = pOT( 5)
  R = pOT( 4 )
  L + R }

pOT( 5)
{ L = pOT(null)
  R = pOT(null )
  return 5 }

pOT( 4 )
{  L = pOT(null)
   R = pOT( null )
   return 4 }

pOT( * )
{ L =pOT( 7)
  R = pOT( 3 )
  L * R }

postOT( 7 )
{  L = pOT( null)
   R = pOT( null )
   return 7 }

postOT( 3 )
{  L = pOT( null)
   R = pOT( null )
   return 3 }
```