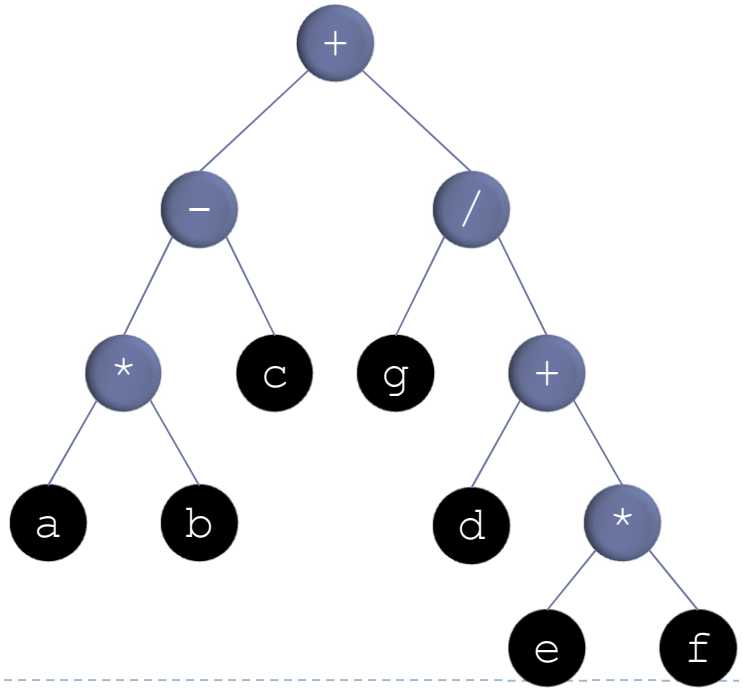
Expression Trees

Expression Tree

It is a special kind of binary tree in which:

- Each leaf node contains a single operand
- Each nonleaf node contains a single binary operator
- The left and right subtrees of an operator represent subexpressions that must be evaluated before applying the operator at the root of the subtree.

- `(a * b - c) + g / (d + e * f)`

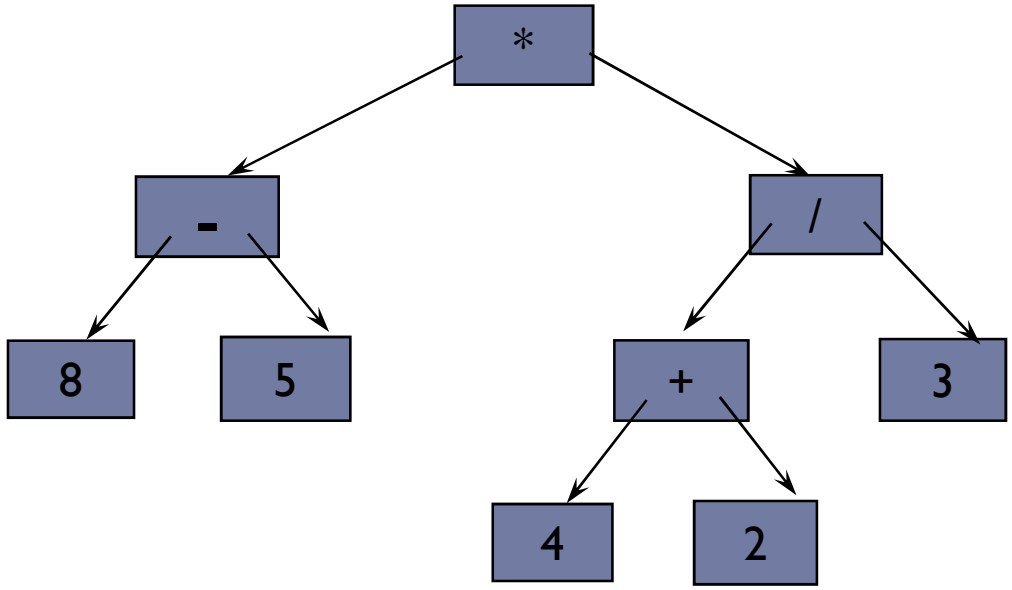


Levels Indicate Precedence

- The levels of the nodes in the tree indicate their relative precedence of evaluation (we do not need parentheses to indicate precedence).
- Operations at higher levels of the tree are evaluated later than those below them.
- The operation at the root is always the last operation performed.

How to generate the infix, prefix, postfix expressions?

- Infix: ((8 - 5) * ((4 + 2) / 3))
- Prefix: * - 8 5 / + 4 2 3
- Postfix: 8 5 - 4 2 + 3 / *



Building an Expression Tree

```
(a * b - c) + g / (d + e * f)
Postfix:   a b * c - g d e f * + / + #
```

```
sym = readSymbol();
while sym ≠ '#'  //'#' denotes delimeter
   if sym == operand
        new=getNode();
        new→data=sym;
        Push(new);
   else
        ptr1=Pop();
        ptr2=Pop();
        new=getNode();
        new→data=sym;
        new→Lchild=ptr2;
        new→Rchild=ptr1;
        Push(new);
   EndIf
   sym = readSymbol();
EndWhile
```

Evaluating an Expression Tree `eet()`

```
ptr = Root;
if ptr ≠ NULL
        Lptr = ptr→Lchild;              Rptr = ptr→Rchild;
        if Lptr→data is an operand
                d1 = Lptr→data;
        else
                d1 = eet(Lptr);
        EndIf

        if Rptr→data is an operand
                d2 = Rptr→data;
        else
                d2 = eet(Rptr);
        EndIf

        operator = ptr→data;
        value = d1 operator d2;
        return(value)
EndIf
```