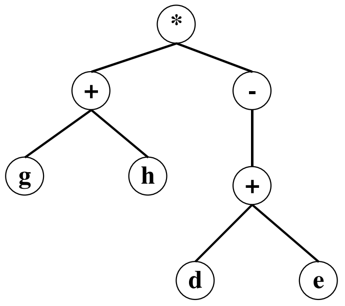
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?

Postfix Order?

Infix Order?



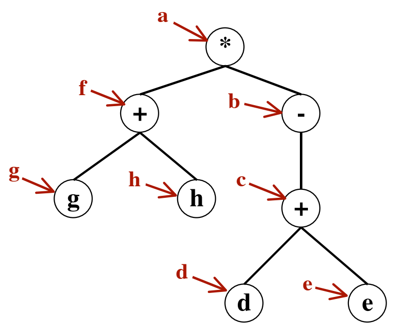
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?

Postfix Order?

Infix Order?



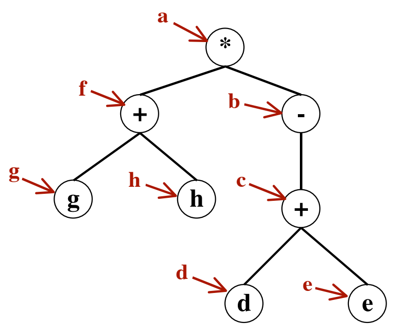
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?
- * + g h - + d e

Postfix Order?

Infix Order?



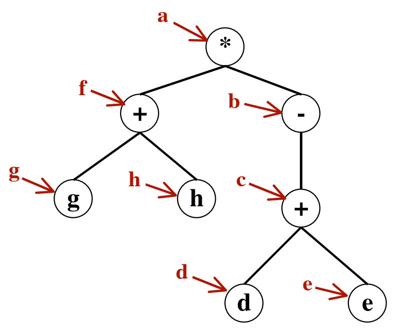
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?
- * + g h - + d e

Postfix Order?
- g h + d e + - *

Infix Order?



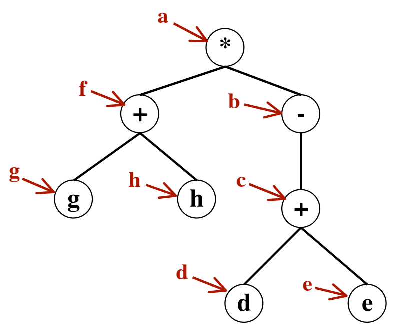
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?
- * + g h - + d e

Postfix Order?
- g h + d e + - *

Infix Order?
- g + h * - d + e



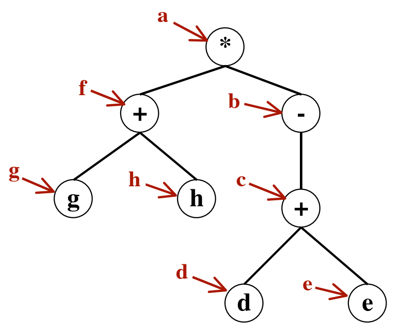
The Tree Assignment

Tree
- Nodes, Edges, Labels, Root
- Binary Nodes
- Unary Nodes
- Traverse the Tree

Prefix Order?
- * + g h - + d e

Postfix Order?
- g h + d e + - *

Infix Order?
- g + h * - d + e
- ((g + h) * (- (d + e)))

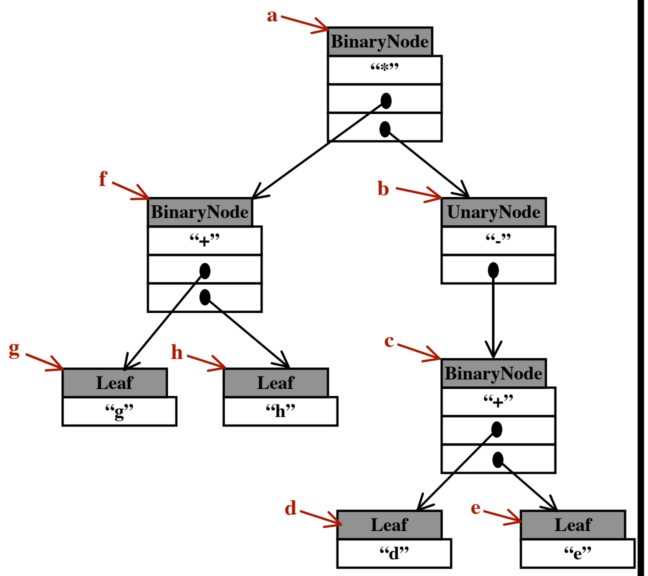


The Tree Assignment

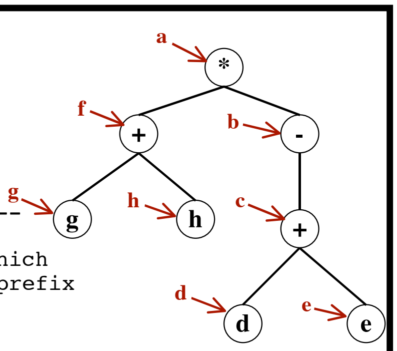
Representation?
- Nodes --> Objects
  - BinaryNodes
  - UnaryNodes
  - Leafs
- Edges --> Pointers

Classes:

```
Node
    Leaf
    BinaryNode
    UnaryNode
```



The Tree Assignment



```
// Tree -- Program to Manipulate Trees
//
// Harry Porter  --  01/03/03
//


//------------------ Tree -------------------
//
// This class contains the "main" method, which
// builds a tree and then prints it out in prefix
// and infix form.
//
class Tree {
    public static void main (String [] args) {
        Node h = new Leaf ("h");
        Node g = new Leaf ("g");
        Node f = new BinaryNode ("+", g, h);
        Node e = new Leaf ("e");
        Node d = new Leaf ("d");
        Node c = new BinaryNode ("+", d, e);
        Node b = new UnaryNode ("-", c);
        Node a = new BinaryNode ("*", f, b);

        System.out.println ("Prefix = " + a.preFix ());
        System.out.println ("Infix = " + a.inFix ());
    }
}
```

The Tree Assignment

```
//------------------------------------- Node -------------------------------------
//
// Each instance of this class represents a node in a tree.  Each node will have a label.
// This class is abstract; no instances of it can be created.  Only the subclasses (Leaf,
// BinaryNode, ...) will be instantiated.
//
abstract class Node {

    //
    // Fields
    //
    String label;


    //
    // Constructor
    //
    Node (String lab) {
        label = lab;
    }


    //
    // preFix () --> String
    //
    // This method returns a String representing the tree in prefix notation.
    //
    abstract String preFix ();


    //
    // inFix () --> String
    //
    // This method returns a String representing the tree in infix notation.
    //
    abstract String inFix ();

}
```

The Tree Assignment

```
//------------------------ Leaf ------------------------
//
// Instances of this class represent leaf nodes, with no children.
//
class Leaf extends Node {

    //
    // Constructor
    //
    Leaf (String n) {
        super (n);
    }

    //
    // preFix () --> String
    //
    // This method returns a String representing this
    // node.
    //
    String inFix () {
        return label;
    }

    //
    // inFix () --> String
    //
    // This method returns a String representing this
    // node.
    //
    String preFix () {
        return label;
    }

}
```

```
//-------------------- BinaryNode -----------------------
//
// Instances of this class represent nodes with two children.
//
class BinaryNode extends Node {

    // Fields
    //
    Node leftChild;
    Node rightChild;

    // Constructor
    //
    BinaryNode (String n, Node left, Node right) {
        super (n);
        leftChild = left;
        rightChild = right;
    }

    // preFix () --> String
    //
    // This method returns a String representing the
    // tree in prefix notation.
    //
    String preFix () {
        return label + " " + leftChild.preFix () + " "
                              + rightChild.preFix ();
    }

    // inFix () --> String
    //
    // This method returns a String representing the
    // tree in infix notation.
    //
    String inFix () {
        return "(" + leftChild.inFix () + " " + label
                      + " " + rightChild.inFix () + ")";
    }

}
```

```
//-------------------- UnaryNode -----------------------
//
// Instances of this class represent nodes with one child.
//
class UnaryNode extends Node {

    //
    // Fields
    //
    Node child;

    //
    // Constructor
    //
    UnaryNode (String n, Node ch) {
        super (n);
        child = ch;
    }

    //
    // preFix () --> String
    //
    // This method returns a String representing the
    // tree in prefix notation.
    //
    String preFix () {
        return label + " " + child.preFix ();
    }

    //
    // inFix () --> String
    //
    // This method returns a String representing the
    // tree in infix notation.
    //
    String inFix () {
        return "(" + label + " " + child.inFix () + ")";
    }

}
```