CS 112 - Data Structures

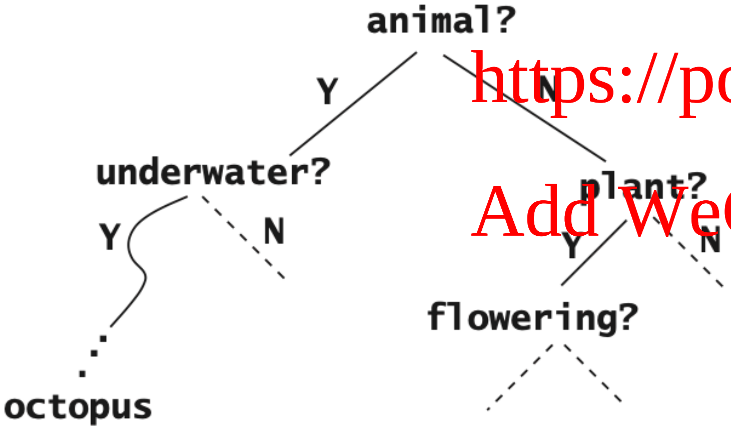
Sesh Venugopal

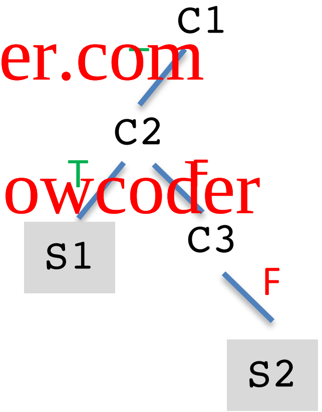
Binary Trees

Binary Trees

BST and AVL Tree are special kinds of binary trees, specialized for searching: every node holds a key that can be searched for

Binary trees can be used for other purposes. For example, you can use a binary tree to model a 20-questions game, or an if statement:





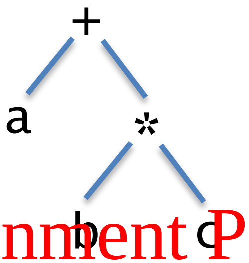
```
if (C1) then
   if (C2) then
      S1
   else
      if (!C3) then
         S2
      endif
   endif
endif
```

A non-search binary tree models some scenario with nodes holding context-specific info (e.g. question, condition) and branches showing at most two outcomes

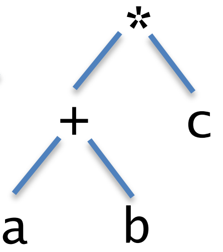
Expression Tree

An important use of a binary tree is to model arithmetic expressions:

a + b * c →



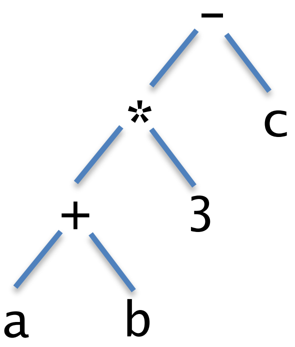
(a + b) * c →



Unary operators are not allowed, only binary operators permitted. Which means every node has either no children (leaf nodes) or exactly two children (internal nodes). No node can have exactly one child.

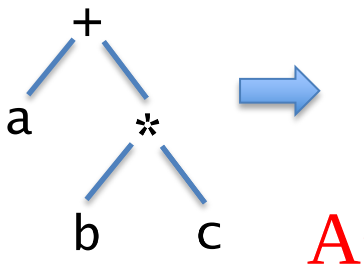
Every internal node is an operator, and every leaf node is an operand

a + b * 3 – c →

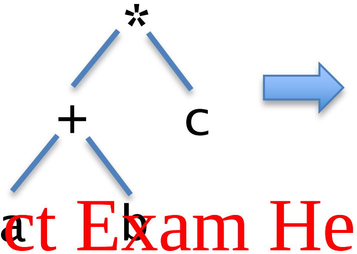


Expression Tree Traversals

Inorder traversal of expression tree
(recurse on Left subtree, then Visit root, then Recurse on right subtree – LVR):



a + b * c

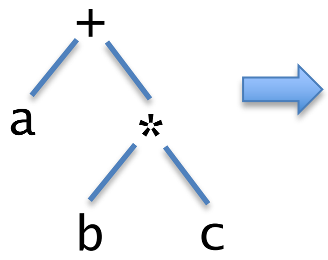


a + b * c

But this is a different expression than the original from which the tree was built!

This ambiguity arises because the expression tree does not store parentheses

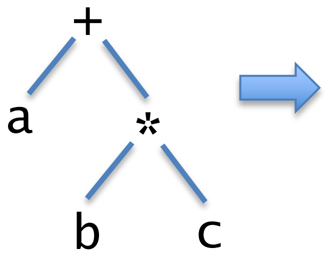
You can use a preorder traversal instead. In a preorder traversal, visit the root first, then recursively traverse the left subtree, then recursively traverse the right subtree (VLR):



+ a * b c

The operator *precedes* the two operands on which it applies

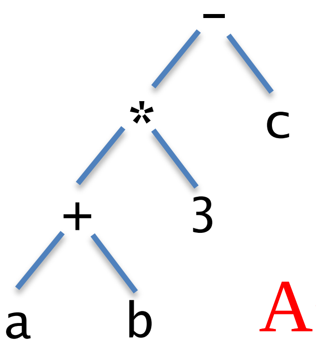
Or, you can use a postorder traversal, in which recursively traverse the left subtree, then recursively traverse the right subtree, and finally visit (LRV):



a b c * +

The operator *follows* the two operands on which it applies

Tree Traversals

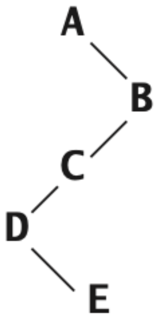


Inorder: a + b * 3 – c Infix form of expression

Preorder: – * + a b 3 c Prefix form

Postorder: a b + 3 * c – Postfix form

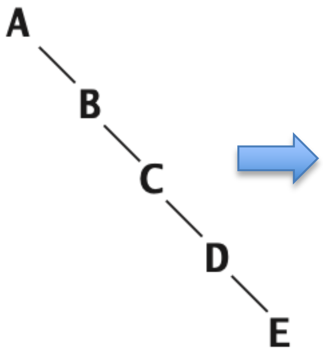
Inorder, preorder, and postorder traversals are doable on ANY binary tree, not just expression trees:



Pre: A B C D E

In: A D E C B

Post: E D C B A



Pre: A B C D E

In: A B C D E

Post: E D C B A

Prefix and Postfix Expressions

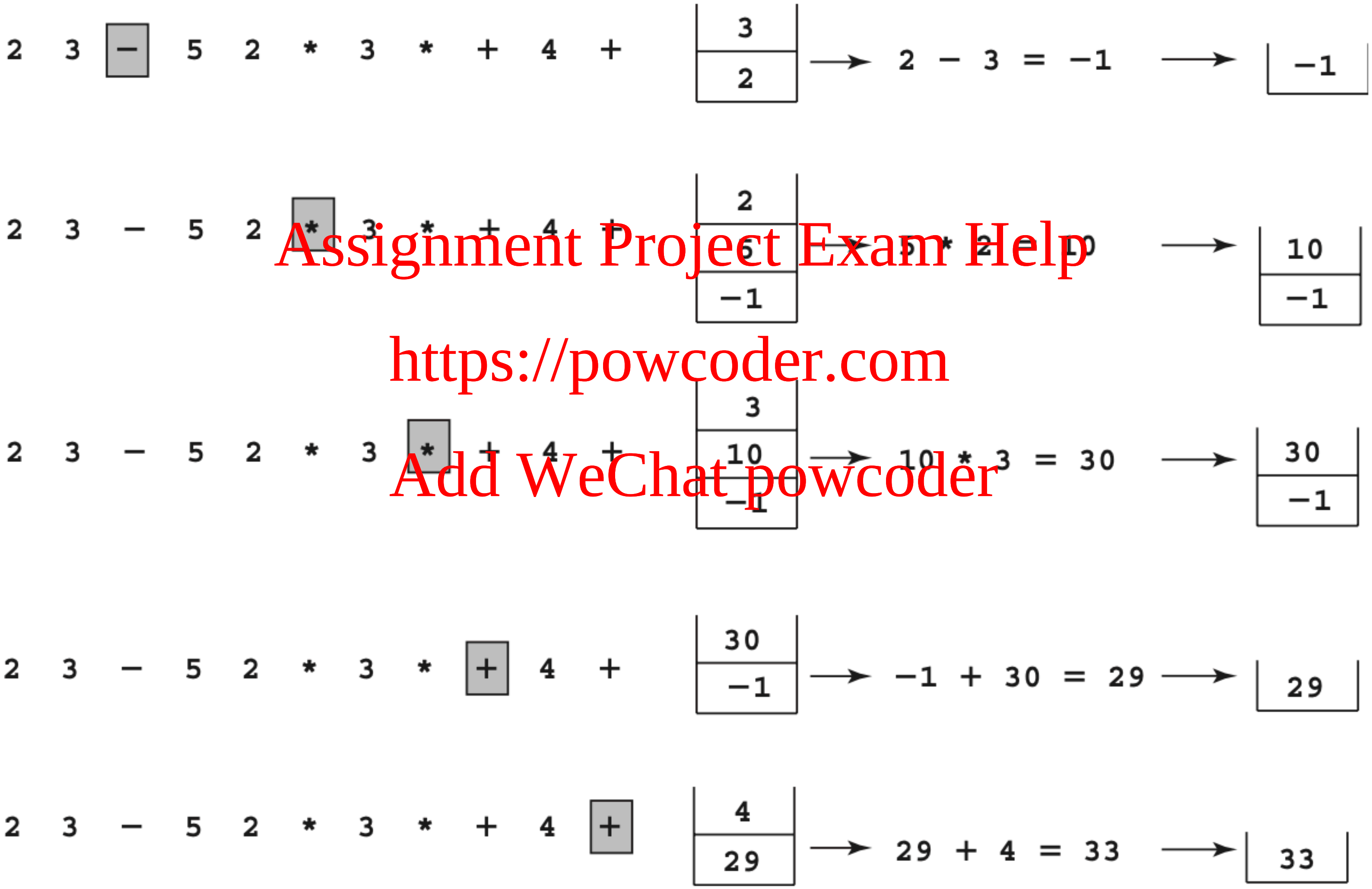
The advantage of prefix and postfix expressions is that you do not need parentheses, ever:

Infix	Prefix	Postfix
a + b * c	+ a * b c	a b c * +
(a + b) * c	* + a b c	a b + c *
(a + b) * (3 – c)	* + a b – 3 c	a b + 3 c – *

The postfix form, in particular, is very easy to evaluate, using a stack: Scan the expression left to right. If an operand/constant is seen, push its value on the stack. If an operator is seen, apply it on the top two operands (in that order), and push the result on to stack. The result is the single value on the stack when the scan is done.

Postfix Expression Evaluation: Example

(Equivalent Infix expression: 2 – 3 + 5 * 2 * 3 + 4)



2 3 $\boxed{-}$ 5 2 * 3 * + 4 +

3
2

→ 2 − 3 = −1 →

−1

2 3 − 5 2 $\boxed{*}$ 3 * + 4 +

2
5
−1

→ 5 * 2 = 10 →

10
−1

2 3 − 5 2 * 3 $\boxed{*}$ + 4 +

3
10
−1

→ 10 * 3 = 30 →

30
−1

2 3 − 5 2 * 3 * $\boxed{+}$ 4 +

30
−1

→ −1 + 30 = 29 →

29

2 3 − 5 2 * 3 * + 4 $\boxed{+}$

4
29

→ 29 + 4 = 33 →

33