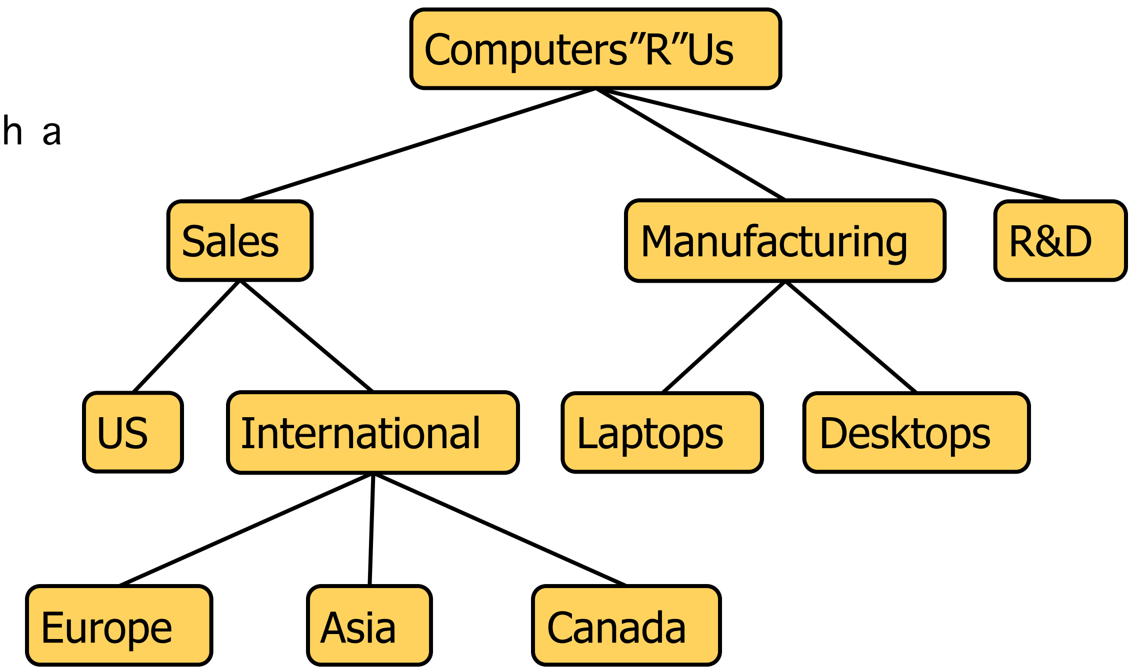
Trees

Algorithms & Data Structures
ITCS 6114/8114

Dr. Dewan Tanvir Ahmed
Department of Computer Science
University of North Carolina at Charlotte

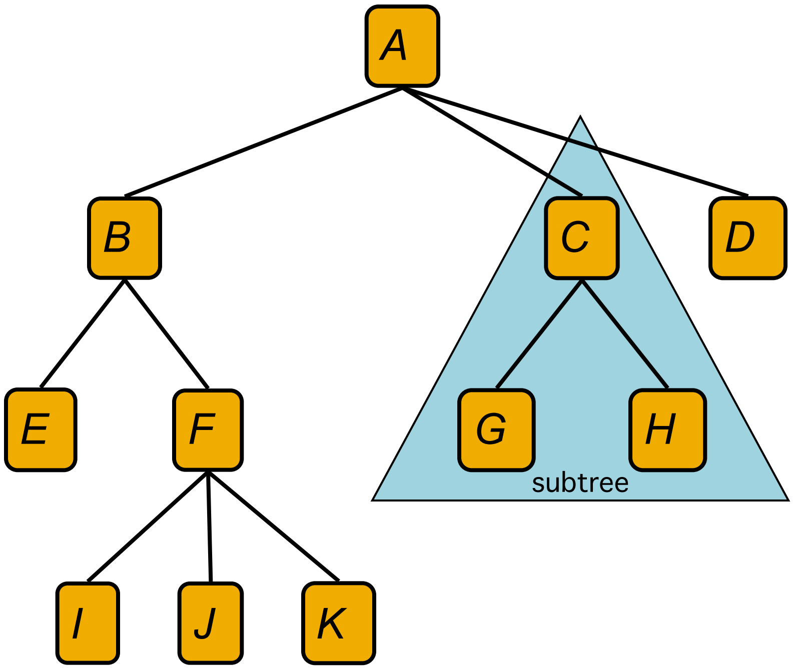
What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants



Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)

- Query methods:
  - `boolean isInternal(p)`
  - `boolean isExternal(p)`
  - `boolean isRoot(p)`
- Update methods:
  - `swapElements(p, q)`
  - `object replaceElement(p, o)`
- Additional update methods may be defined by data structures implementing the Tree ADT

Depth and Height

- Let v be a node of a Tree T. The depth of v is the number of ancestor of v, excluding v itself.

$$depth(v) = \begin{cases} 0 & if\ v\ is\ root \\ 1 + depth(u) & where\ u\ is\ the\ parent\ of\ v \end{cases}$$

```
Algorithm depth (T, v)
    if T.isRoot(v) then
        return 0
    else
        return 1 + depth(T,T.partent(v))
```

Depth and Height

- Running time of depth (T, v) is $O(1 + d_v)$
  - As it performs a constant-time recursive steps for each ancestor of v

    d_v = the depth of the node in T
  - Worst case: $O(n)$

Depth and Height

- The height of tree T is the maximum depth of an external node of T.
- If you apply the above depth-finding algorithm to each node in T, the running time to compute the height of T be $O(n^2)$

Depth and Height

$$height(v) = \begin{cases} 0 & \text{if } v \text{ is leaf} \\ 1 + \text{max height of its children} & \text{else} \end{cases}$$

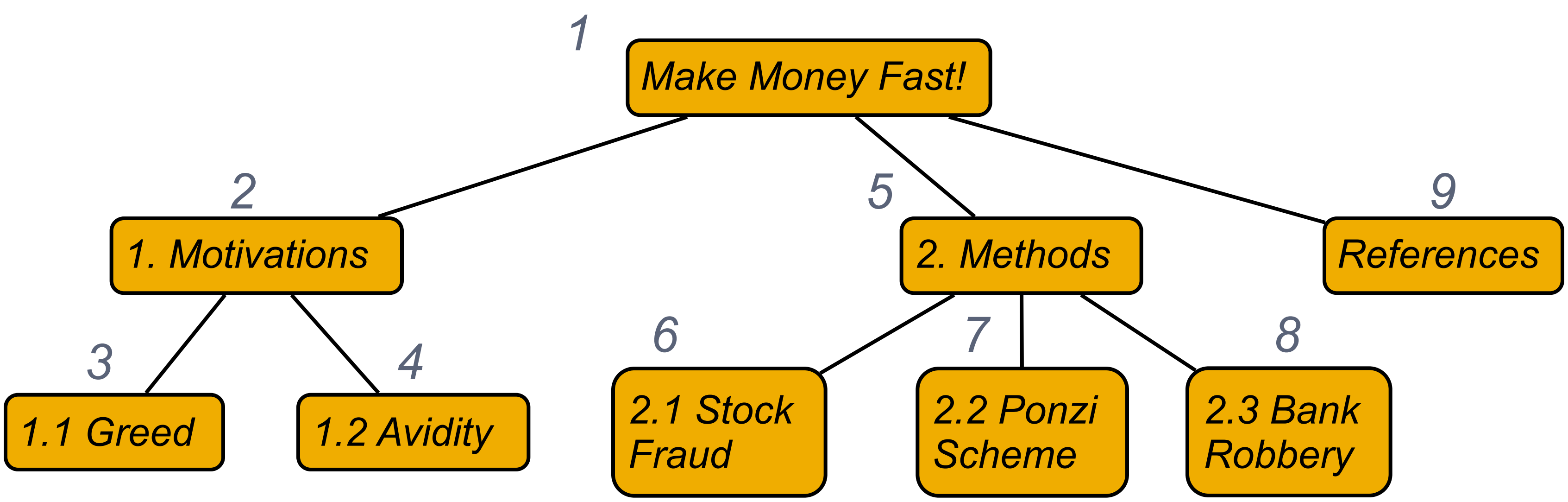
- The height of a tree T is the height of the root of T

```
Algorithm height (T, v)
    if T.isExternal(v) then
        return 0
    else
        h = 0
        for each w ∈ T.children(v) do
            h = max(h, height(T, w))
        return 1 + h
```

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

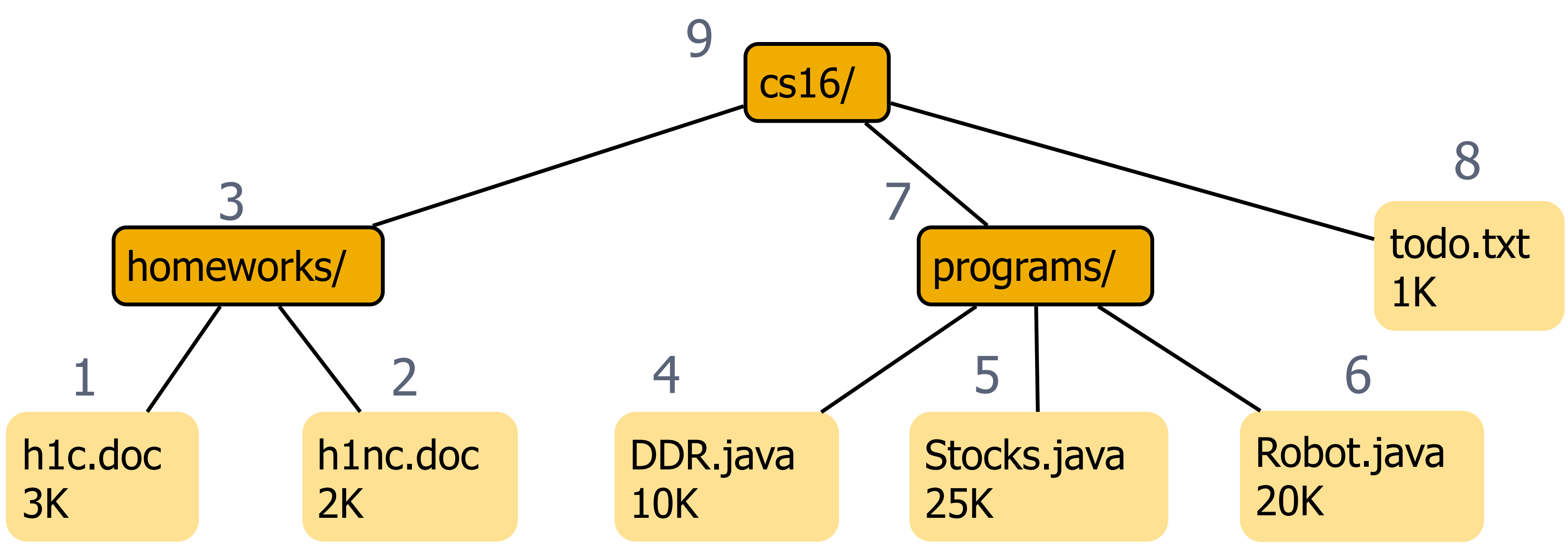
```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preorder (w)
```



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

```
Algorithm postOrder(v)
    for each child w of v
        postOrder (w)
    visit(v)
```

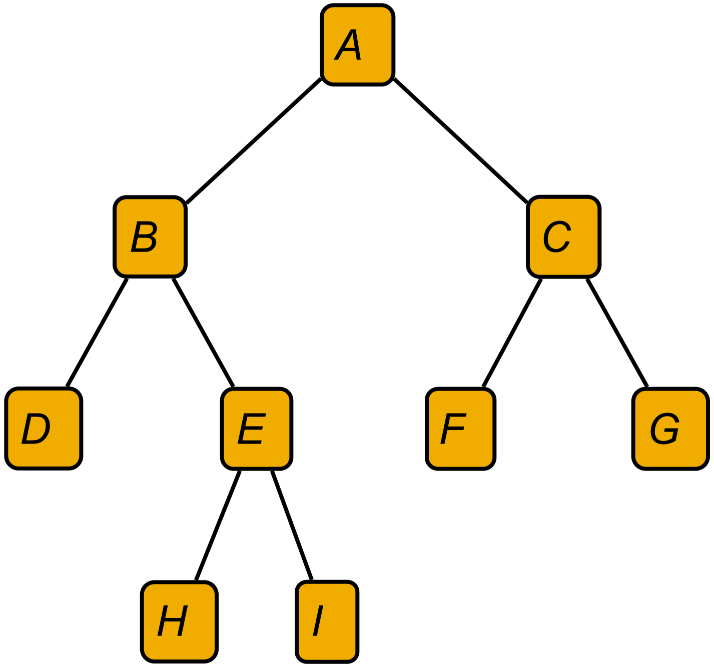


Binary Tree

- A binary tree is a tree with the following properties:
  - Each internal node has two children
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

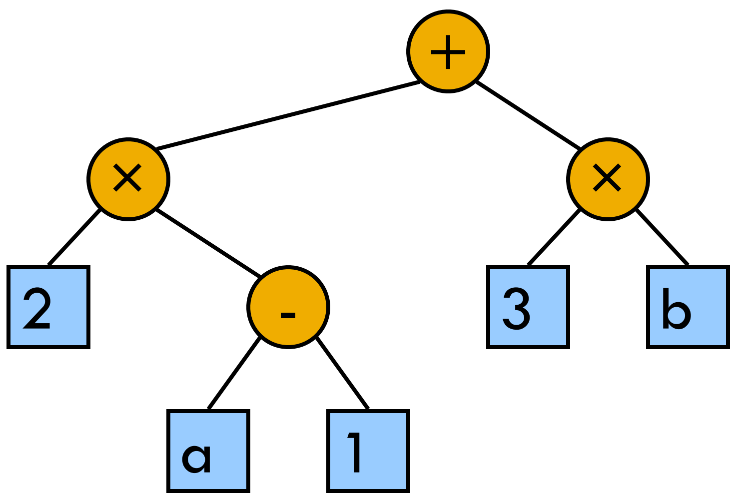
Applications:

- arithmetic expressions
- decision processes
- searching



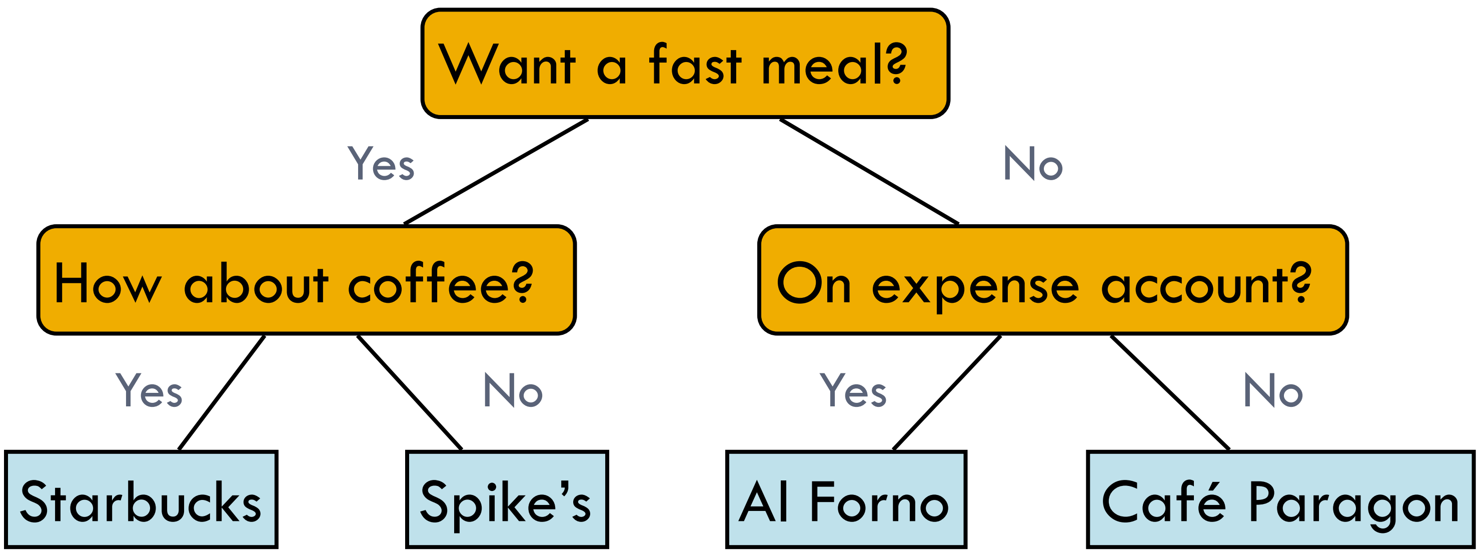
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



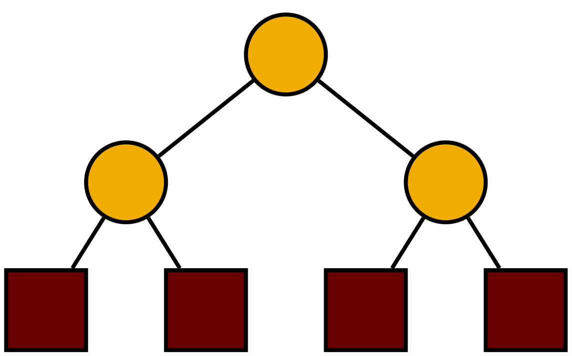
Properties of Binary Trees

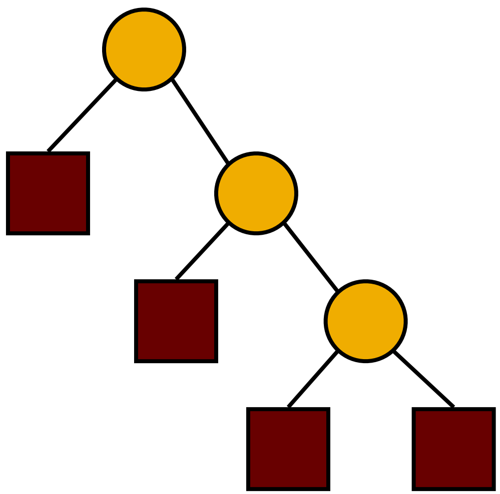
- Notation

n	number of nodes
e	number of external nodes
i	number of internal nodes
h	height

Properties:

- $e = i + 1$
- $n = 2e - 1$
- $h \le i$
- $h \le (n - 1)/2$
- $e \le 2^h$
- $h \ge \log_2 e$
- $h \ge \log_2 (n + 1) - 1$





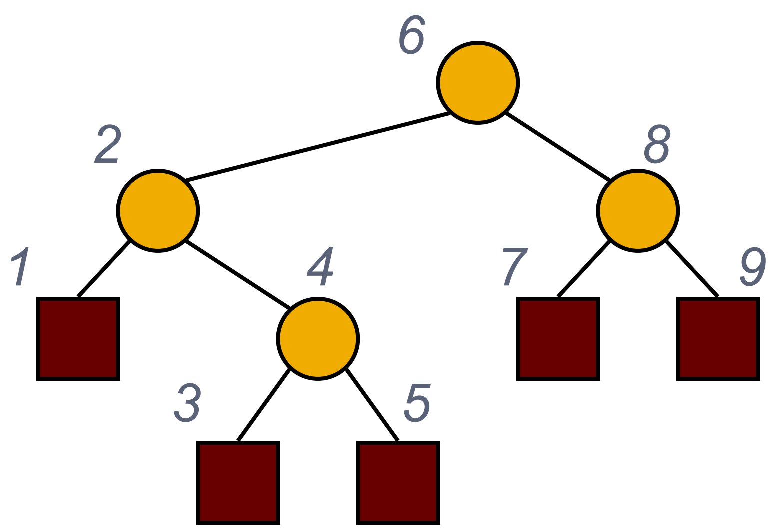
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)

Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

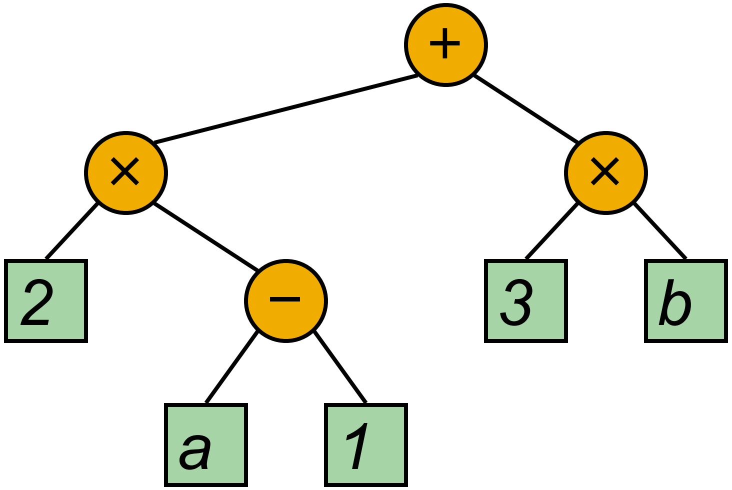
```
Algorithm inOrder(v)
    if isInternal (v)
        inOrder (leftChild (v))
    visit(v)
    if isInternal (v)
        inOrder (rightChild (v))
```



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

```
Algorithm printExpression(v)
    if isInternal (v)
        print("(")
        inOrder (leftChild (v))
    print(v.element ())
    if isInternal (v)
        inOrder (rightChild (v))
        print (")")
```

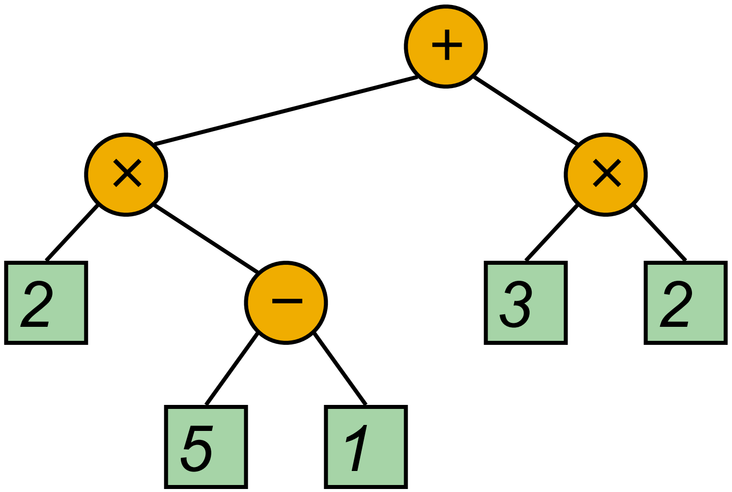


$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

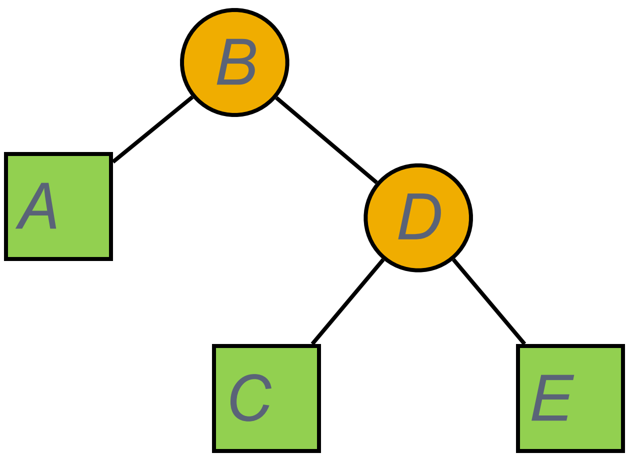
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

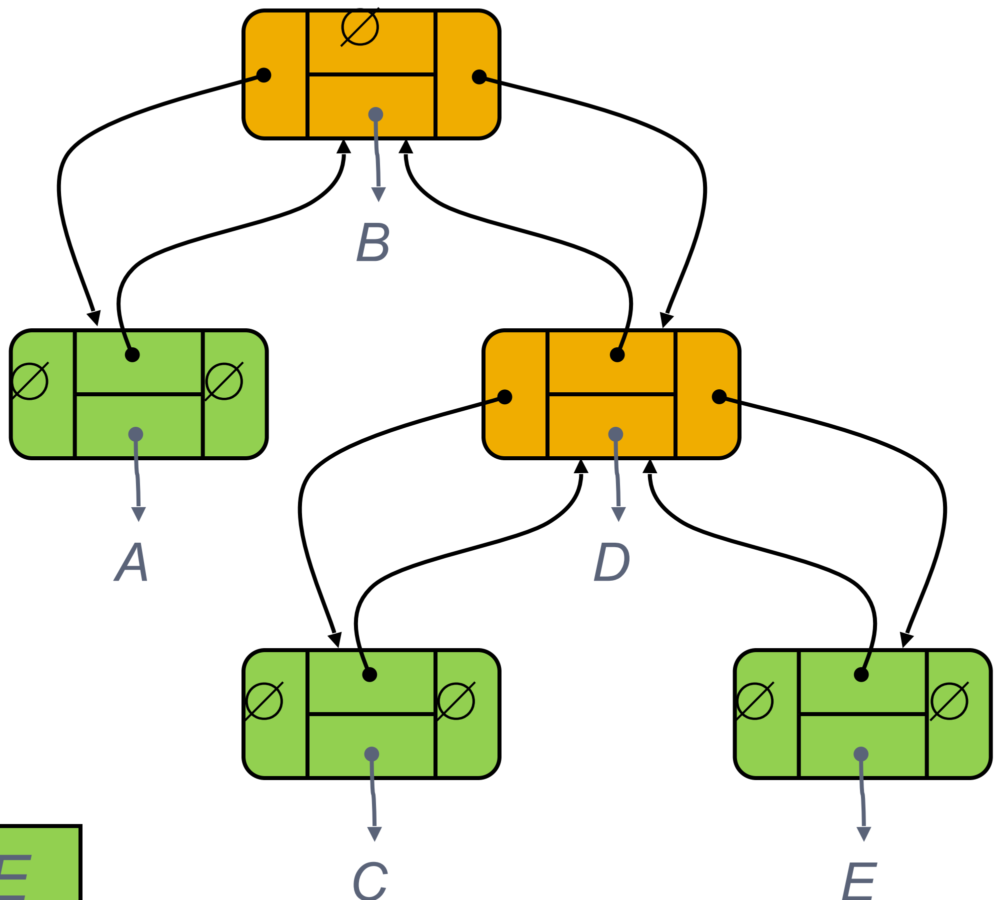
```
Algorithm evalExpr(v)
    if isExternal (v)
        return v.element ()
    else
        x ← evalExpr(leftChild (v))
        y ← evalExpr(rightChild (v))
        ◊ ← operator stored at v
        return x ◊ y
```



Data Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





Reference

- Algorithm Design: Foundations, Analysis, and Internet Examples. Michael T. Goodrich and Roberto Tamassia. John Wiley & Sons.
- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.