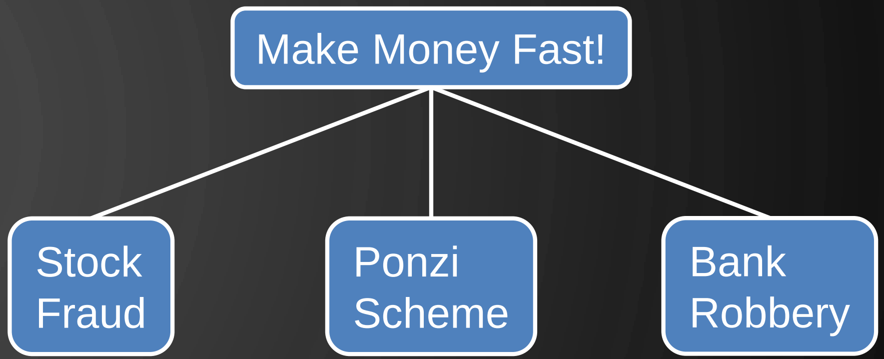


CH 7. TREES

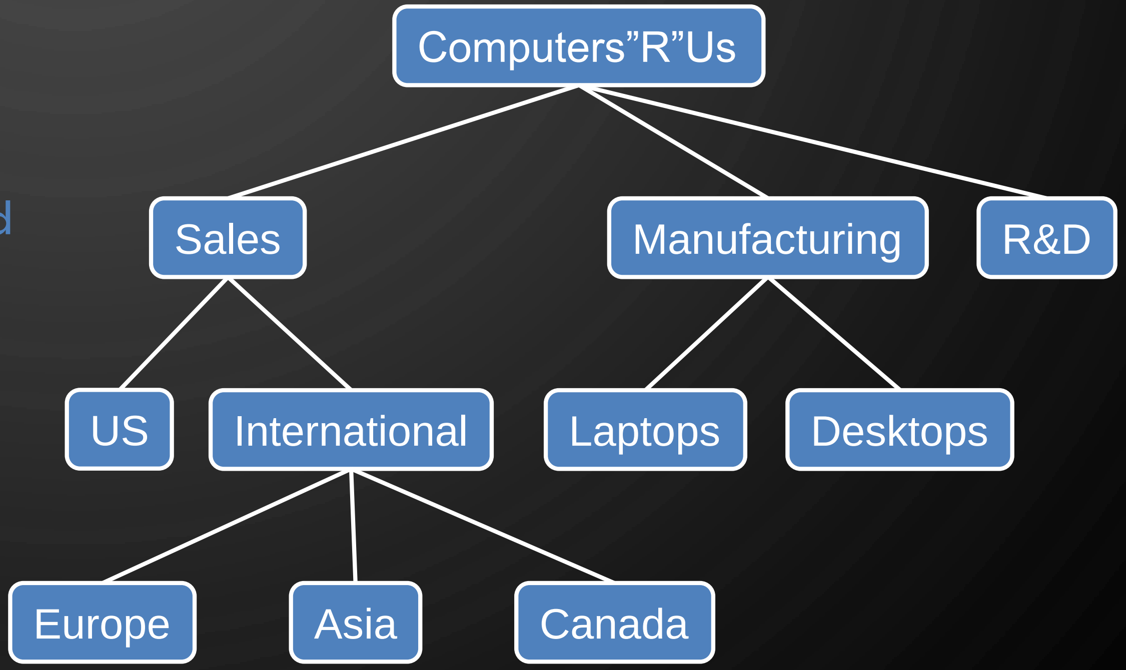
ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN C++, GOODRICH, TAMASSIA AND MOUNT (WILEY 2004) AND SLIDES FROM NANCY M. AMATO

OUTLINE AND READING

- General Trees (Ch. 7.1)
- Tree Traversals (Ch. 7.2)
- Binary Trees (Ch. 7.3)

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

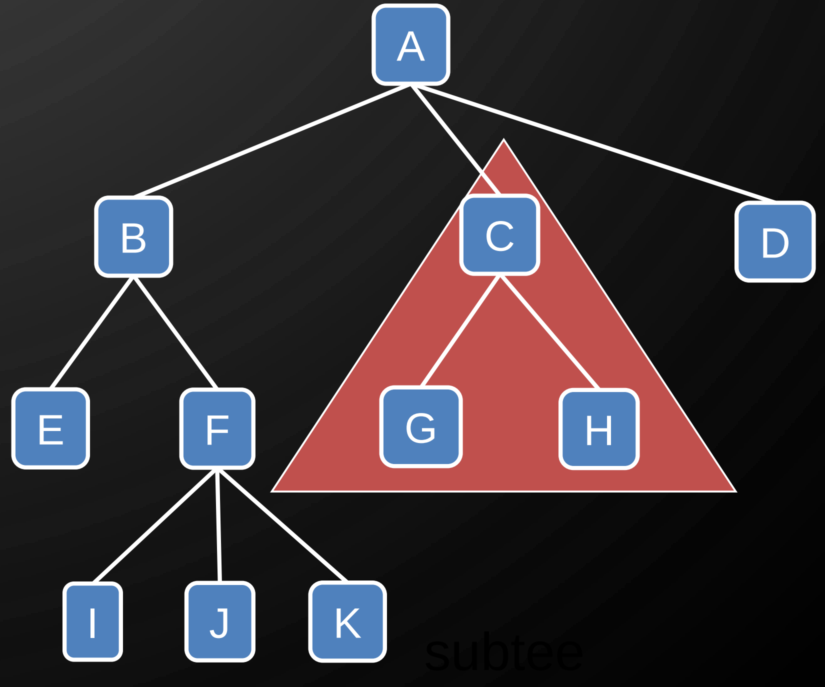


FORMAL DEFINITION

- A tree T is a set of nodes storing elements in a parent-child relationship with the following properties:
  - If T is nonempty, it has a special node called the root of T, that has no parent
  - Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w
- Note that trees can be empty and can be defined recursively!
- Note each node can have zero or more children

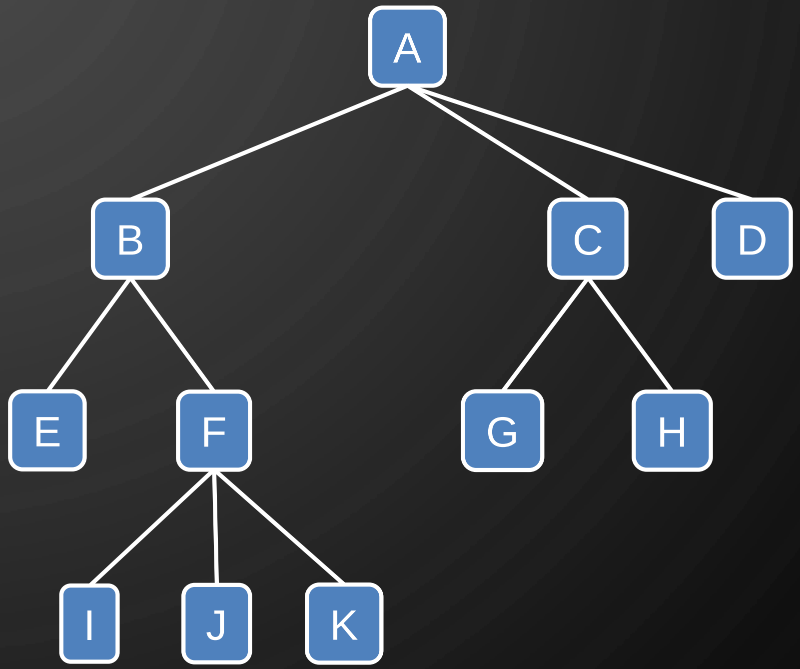
TREE TERMINOLOGY

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- Leaf (aka External node): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, great-grandparent, etc.
- Siblings of a node: Any node which shares a parent
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, great-grandchild, etc.
- Subtree: tree consisting of a node and its descendants
- Edge: a pair of nodes (u, v) such that u is a parent of v $((C, H))$
- Path: A sequence of nodes such that any two consecutives nodes form an edge(A, B, F, J)
- A tree is ordered when there is a linear ordering defined for the children of each node



EXERCISE

- Answer the following questions about the tree shown on the right:
  - What is the size of the tree (number of nodes)?
  - Classify each node of the tree as a root, leaf, or internal node
  - List the ancestors of nodes B, F, G, and A. Which are the parents?
  - List the descendants of nodes B, F, G, and A. Which are the children?
  - List the depths of nodes B, F, G, and A.
  - What is the height of the tree?
  - Draw the subtrees that are rooted at node F and at node K.

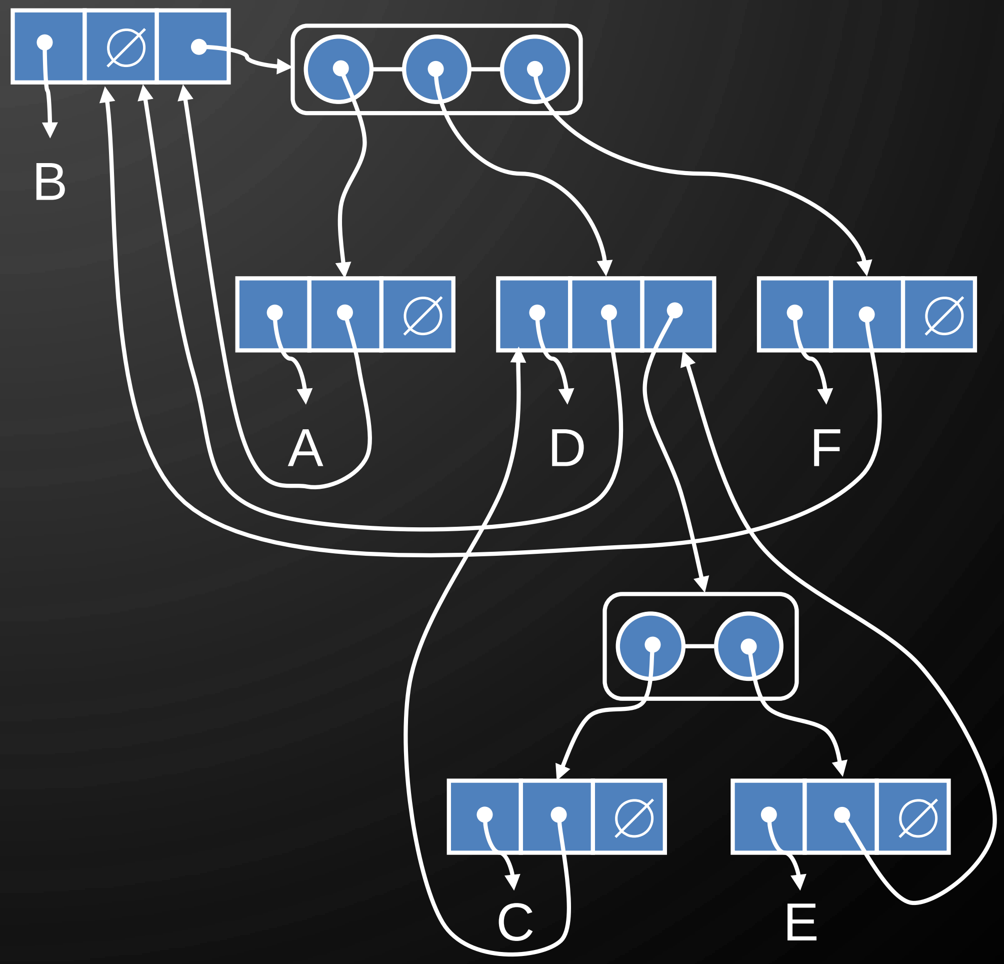


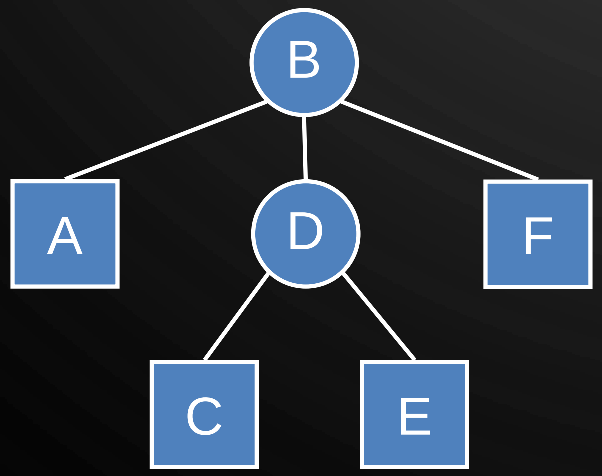
TREE ADT

- We use positions to abstract nodes, as we don't want to expose the internals of our structure
- Position functions:
  - p.parent() – return parent
  - p.children() – list of children positions
  - p.isRoot()
  - p.isLeaf()
- Tree functions:
  - size()
  - empty()
  - root() – return position for root
  - positions() – return list of all positions
- Additional functions may be defined by data structures implementing the Tree ADT, e.g., begin() and end()

A LINKED STRUCTURE FOR GENERAL TREES

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT



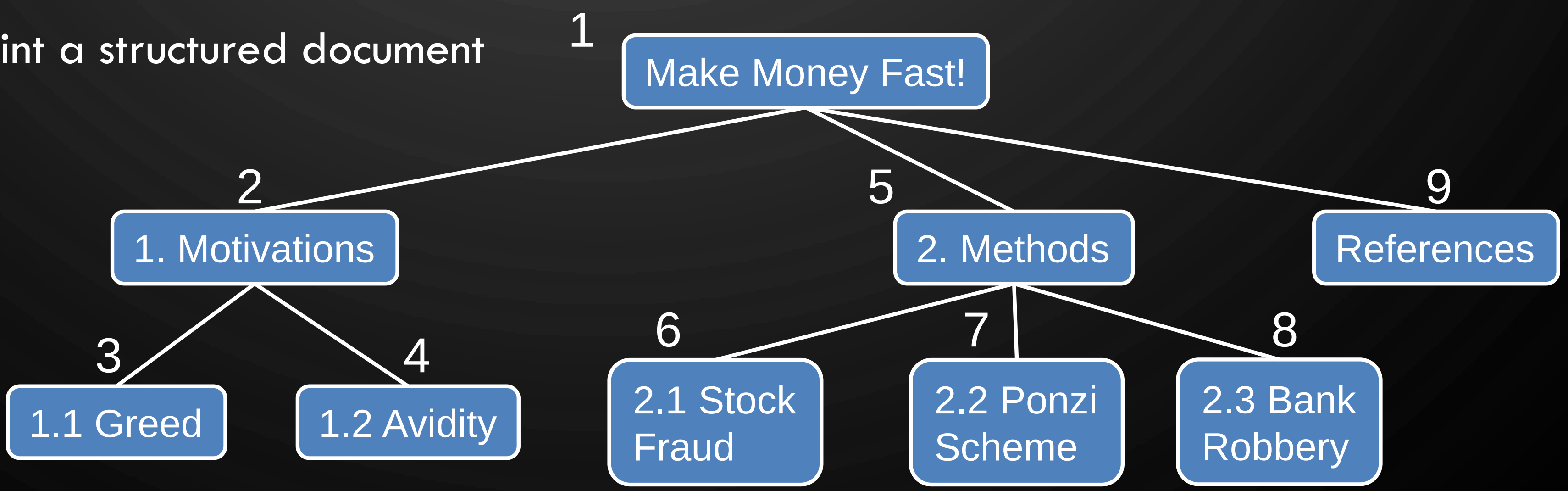


PREORDER TRAVERSAL

- 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

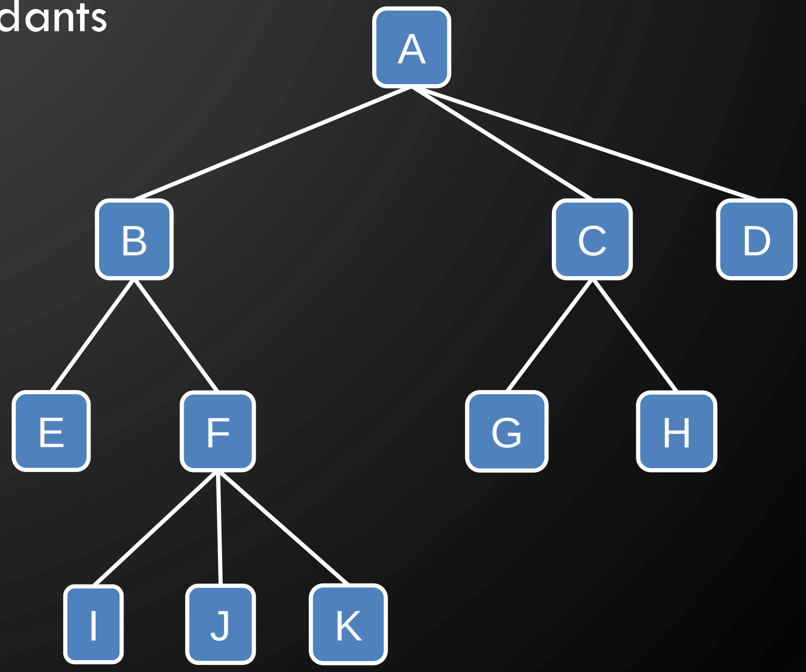
<u>Algorithm preOrder(v)</u>

1. visit(v)
2. **for each** child w of v
3. preOrder(w)



EXERCISE: PREORDER TRAVERSAL

- In a *preorder traversal*, a node is visited before its descendants
- List the nodes of this tree in preorder traversal order.



<u>Algorithm preOrder(v)</u>

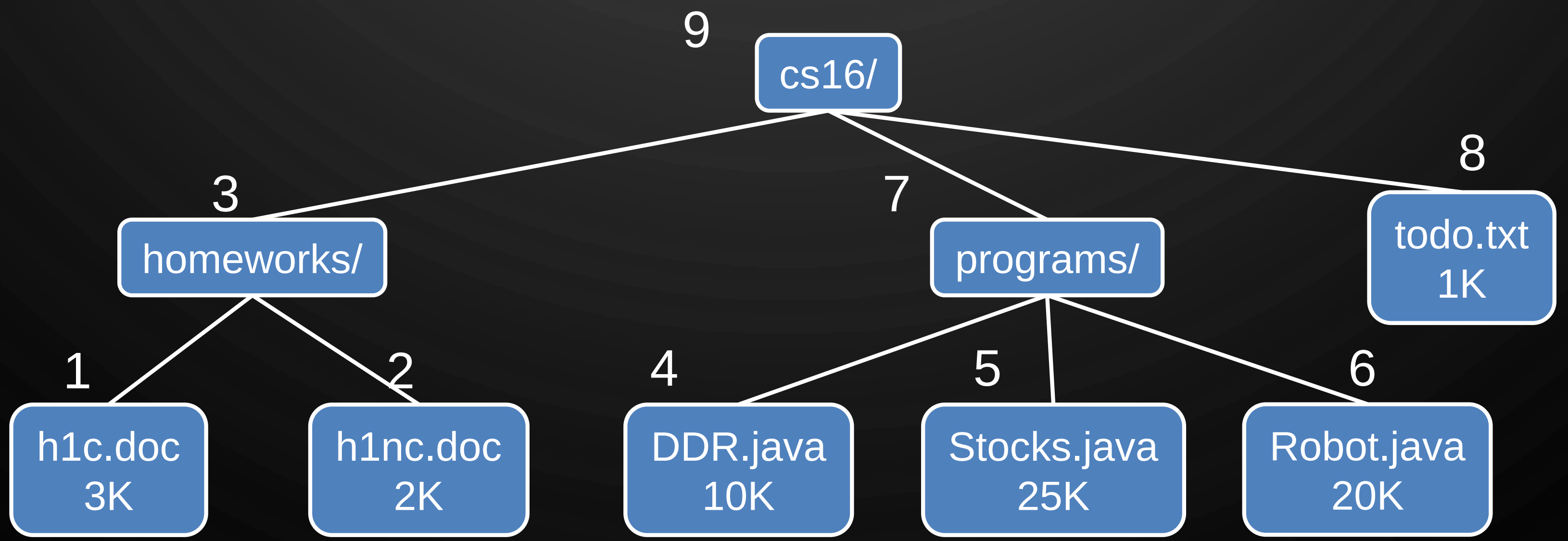
1. visit(v)
2. **for each** child w of v
3. preOrder(w)

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

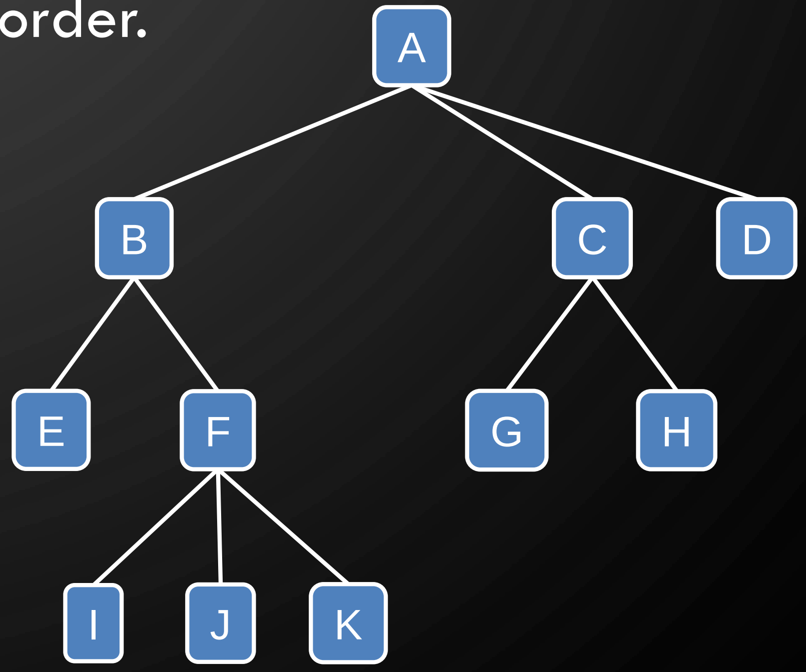
<u>Algorithm postOrder(v)</u>

1. for each child w of v
2.   postOrder(w)
3. visit(v)



EXERCISE: POSTORDER TRAVERSAL

- In a *postorder traversal*, a node is visited after its descendants
- List the nodes of this tree in postorder traversal order.

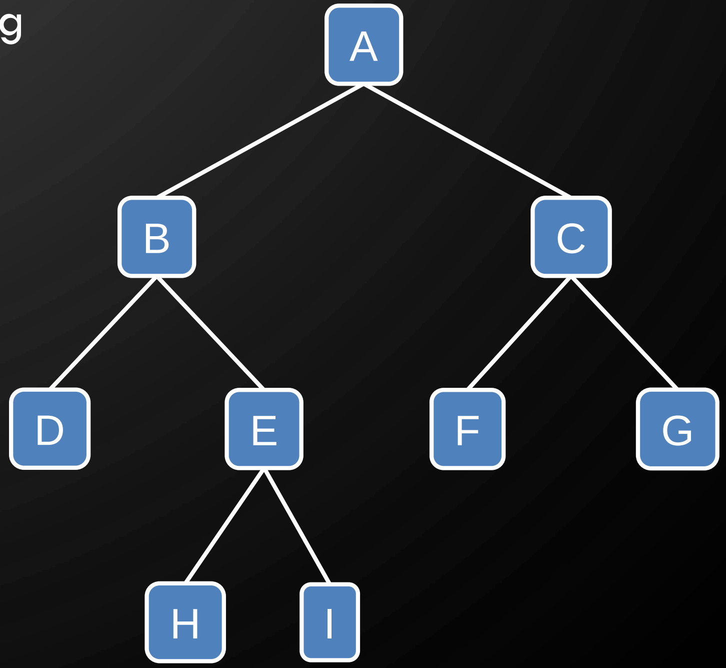


<u>Algorithm postOrder(v)</u>
1. for each child w of v
2.   postOrder(w)
3. visit(v)

BINARY TREE

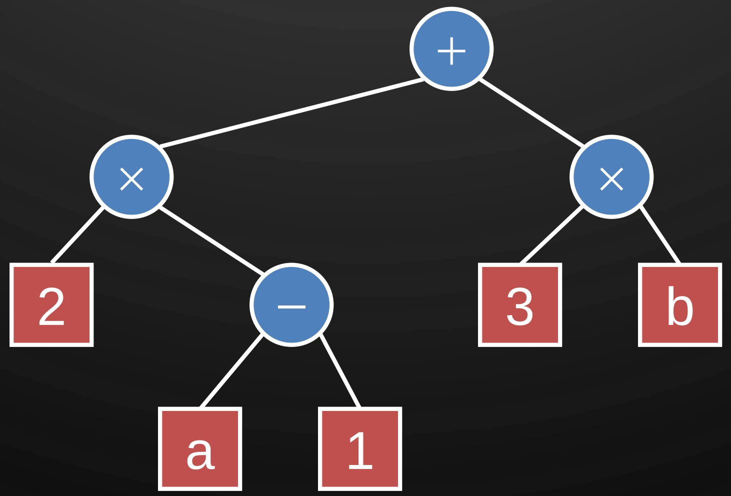
- A *binary tree* is a tree with the following properties:
  - Each internal node has two children
  - The children of a node are an ordered pair
- We call the children of an internal node *left child* and *right child*
- If a child has only one child, the tree is *improper*
- Alternative recursive definition: 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



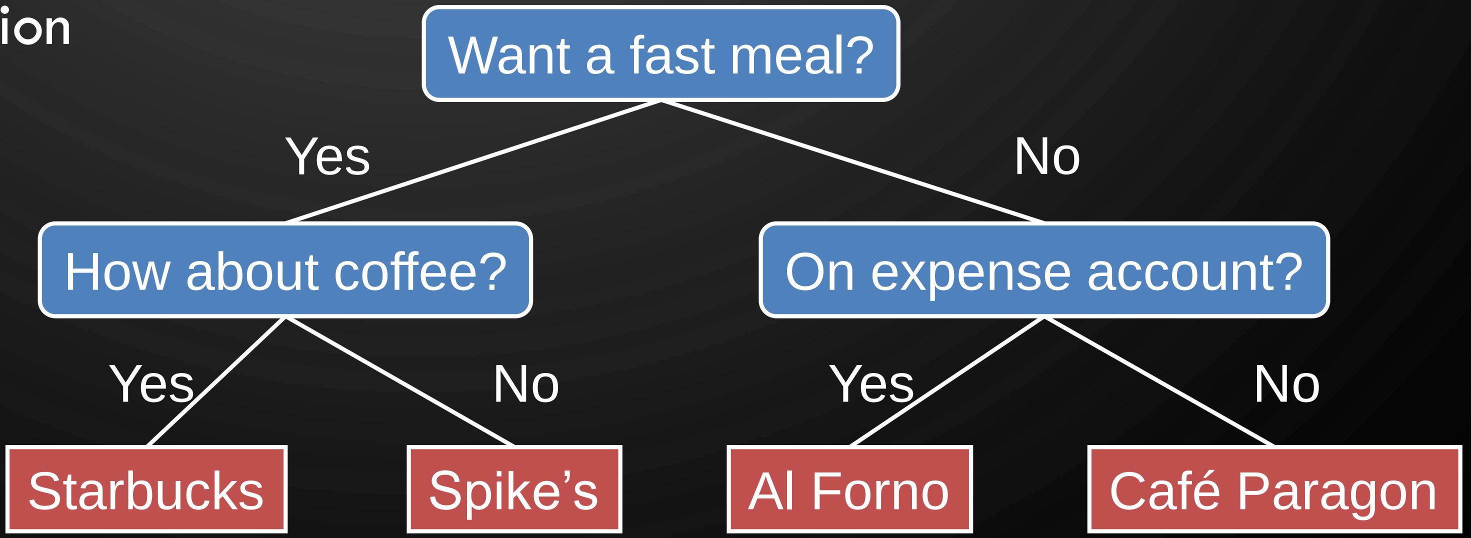
ARITHMETIC EXPRESSION TREE

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



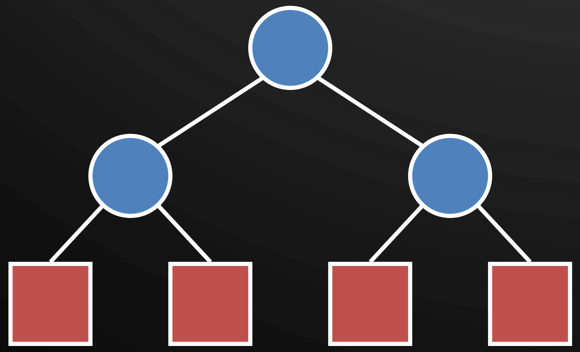
DECISION TREE

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

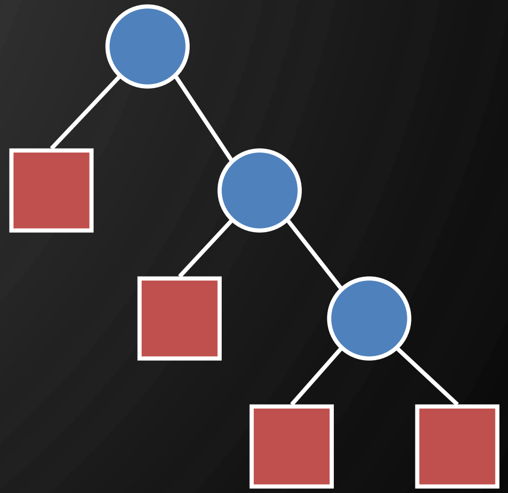


PROPERTIES OF BINARY TREES

- Notation
  - n number of nodes
  - l number of leaves
  - i number of internal nodes
  - h height



- Properties:
  - $l = i + 1$
  - $n = 2l - 1$
  - $h \leq i$
  - $h \leq \frac{n-1}{2}$
  - $l \leq 2^h$
  - $h \geq \log_2 l$
  - $h \geq \log_2(n + 1) - 1$

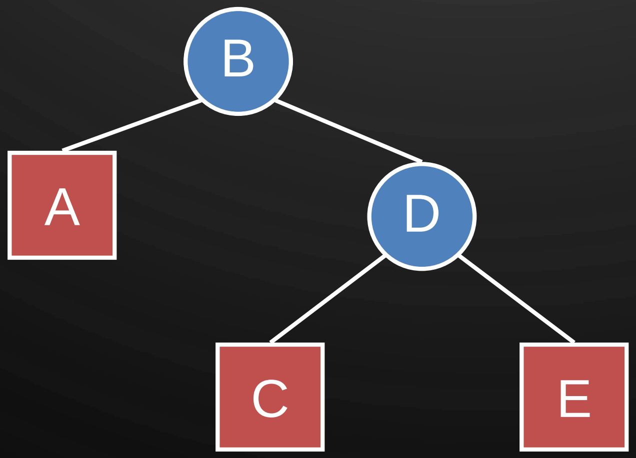


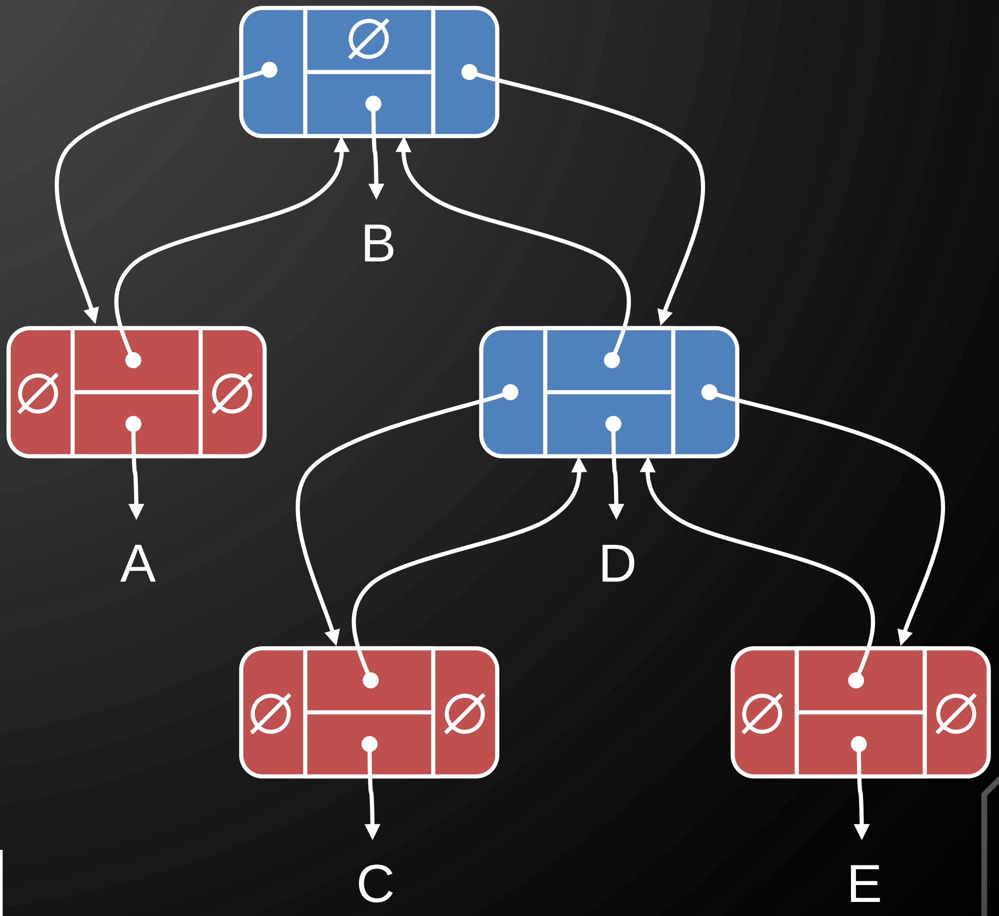
BINARY TREE ADT

- The Binary Tree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional position methods:
  - $p.\mathrm{left}()$
  - $p.\mathrm{right}()$
- Update methods may also be defined by data structures implementing the Binary Tree ADT

A LINKED STRUCTURE FOR BINARY TREES

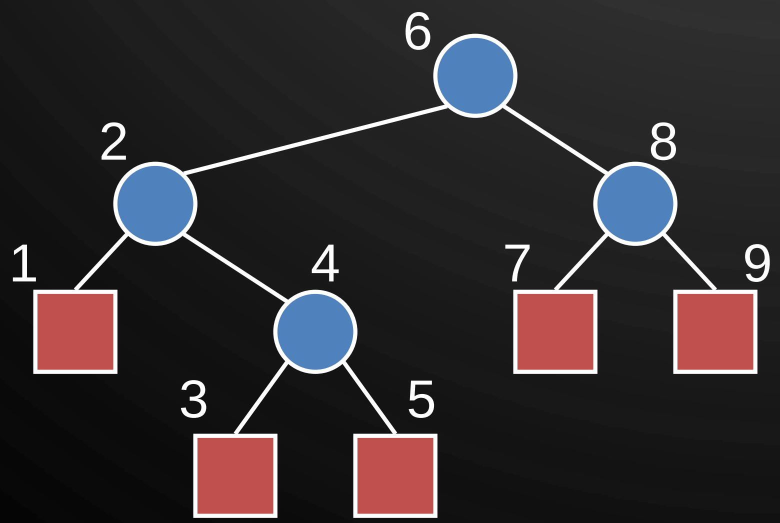
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node





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

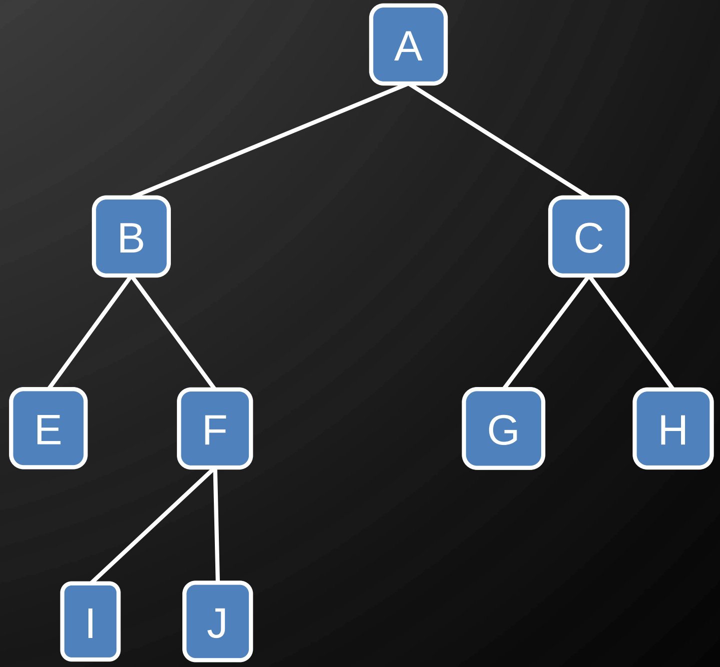


<u>Algorithm $\text{inOrder}(v)$</u>

1. if $v.\text{isInternal}()$
2. $\quad$ $\text{inOrder}(v.\text{left}())$
3. $\text{visit}(v)$
4. if $v.\text{isInternal}()$
5. $\quad$ $\text{inOrder}(v.\text{right}())$

EXERCISE: INORDER TRAVERSAL

- In an *inorder traversal* a node is visited after its left subtree and before its right subtree
- List the nodes of this tree in inorder traversal order.



<u>Algorithm inOrder(v)</u>

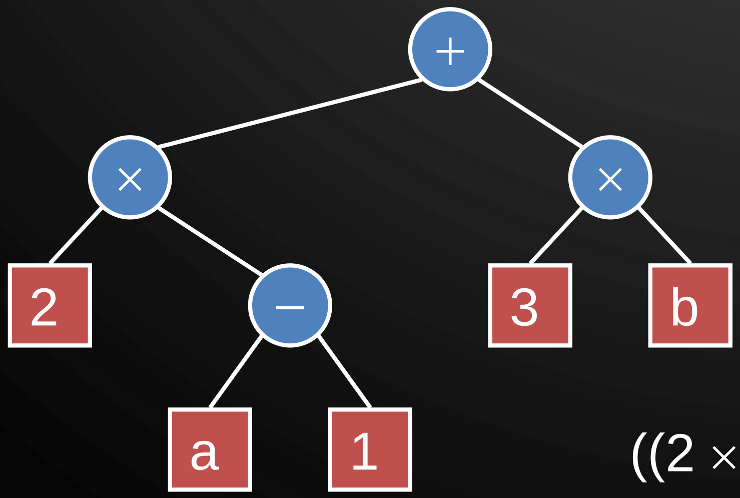
if $v.$isInternal()
    inOrder($v.$left())
visit(v)
if $v.$isInternal()
    inOrder($v.$right())

EXERCISE: PREORDER & INORDER TRAVERSAL

- Draw a (single) binary tree T, such that
  - Each internal node of T stores a single character
  - A preorder traversal of T yields EXAMFUN
  - An inorder traversal of T yields MAFXUEN

APPLICATION
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



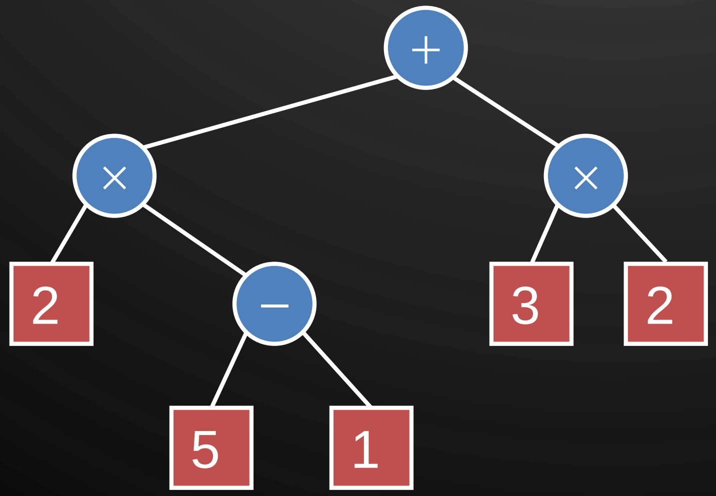
$((2 \times (a - 1)) + (3 \times b))$

Algorithm printExpression(v)

1. **if** $v.\mathrm{isInternal}()$
2.   print("(")
3.   printExpression($v.\mathrm{left}()$)
4. print($v.\mathrm{element}()$)
5. **if** $v.\mathrm{isInternal}()$
6.   printExpression($v.\mathrm{right}()$)
7.   print(")")

APPLICATION
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)
1. if $v.\text{isExternal}()$
2. return $v.\text{element}()$
3. $x \leftarrow \text{evalExpr}(v.\text{left}())$
4. $y \leftarrow \text{evalExpr}(v.\text{right}())$
5. $\circ \leftarrow$ operator stored at v
6. return $x \circ y$

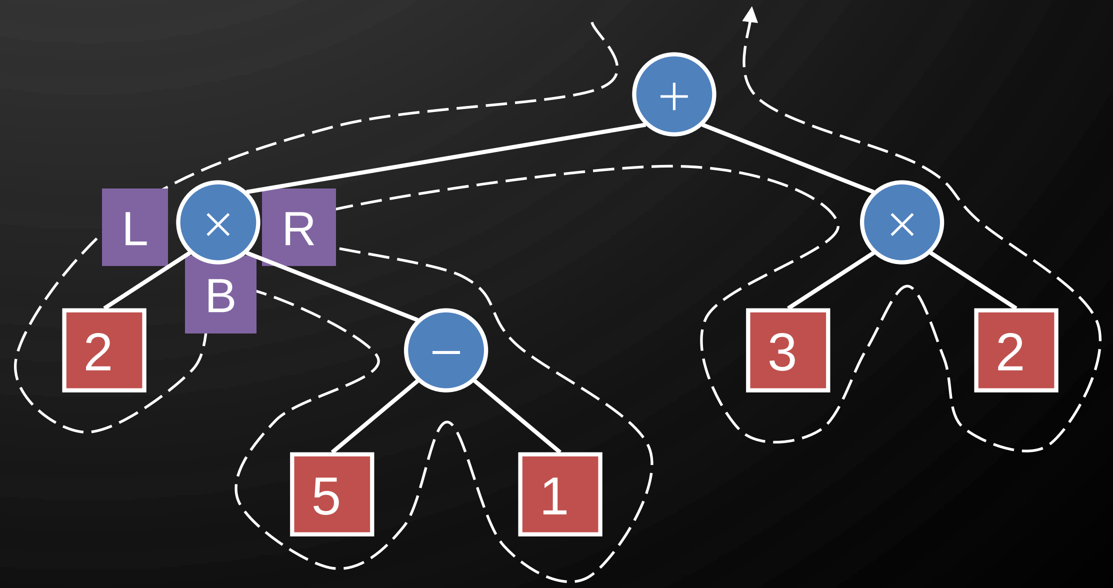
EXERCISE
ARITHMETIC EXPRESSIONS

- Draw an expression tree that has
  - Four leaves, storing the values 1, 5, 6, and 7
  - 3 internal nodes, storing operations +, -, *, /

    operators can be used more than once, but each internal node stores only one
  - The value of the root is 21

EULER TOUR TRAVERSAL

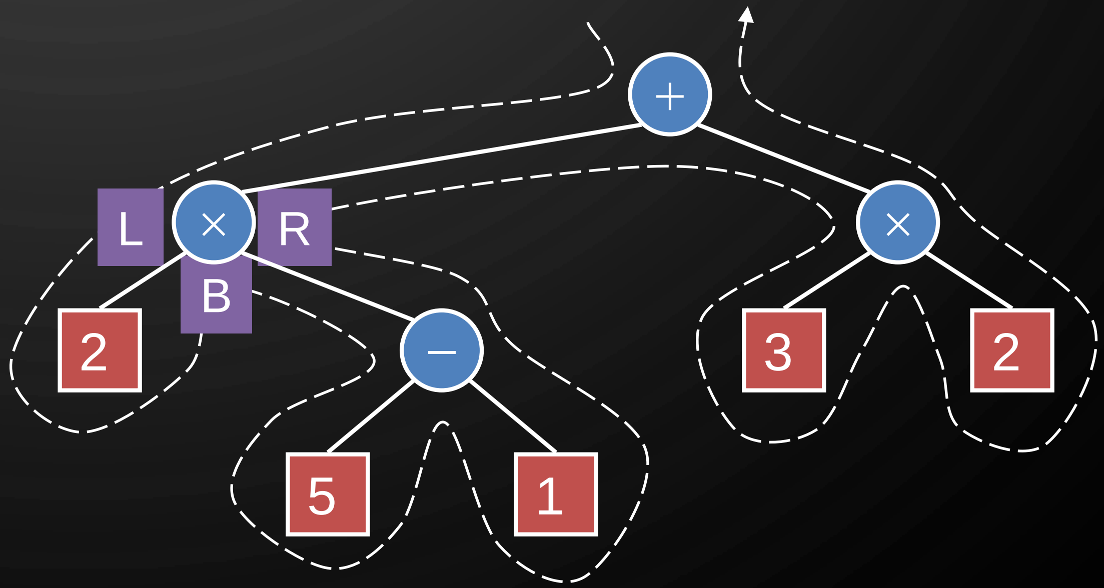
- Generic traversal of a binary tree
- Includes as special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



EULER TOUR TRAVERSAL

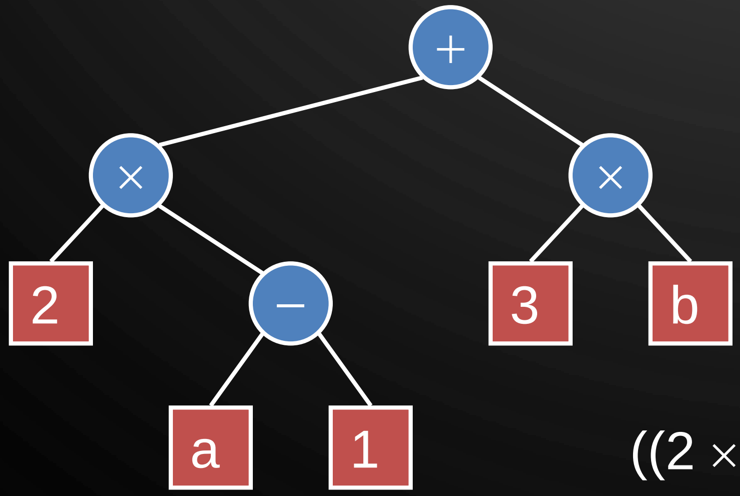
<u>Algorithm eulerTour(v)</u>

1. left_visit(v)
2. **if** v. isInternal()
3. eulerTour(v. left())
4. bottom_visit(v)
5. **if** v. isInternal()
6. eulerTour(v. right())
7. right_visit(v)



APPLICATION
PRINT ARITHMETIC EXPRESSIONS

- Specialization of an Euler Tour traversal
  - Left-visit: if node is internal, print "("
  - Bottom-visit: print value or operator stored at node
  - Right-visit: if node is internal, print ")"



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

<u>Algorithm printExpression(v)</u>

```
if v.isExternal()
    print v.element()
else
    print "("
    printExpression(v.left())
    print operator at v
    printExpression(v.right())
    print ")"
```

INTERVIEW QUESTION 1

- Implement a function to check if a binary tree is balanced. For the purposes of this question, a balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.

GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.

INTERVIEW QUESTION 2

- Given a binary tree, design an algorithm which creates a linked list of all the nodes at each depth (e.g. , if you have a tree with depth D, you'll have D linked lists).

GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.