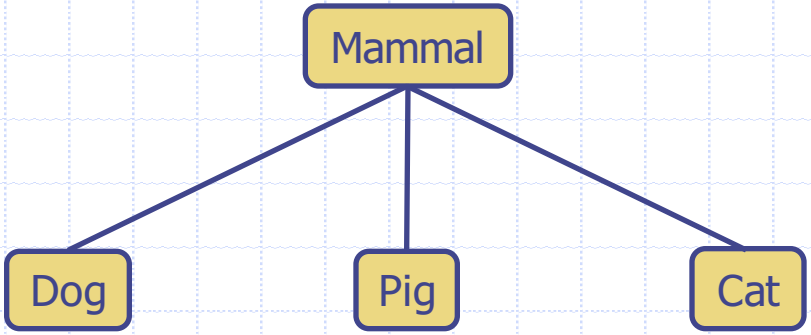
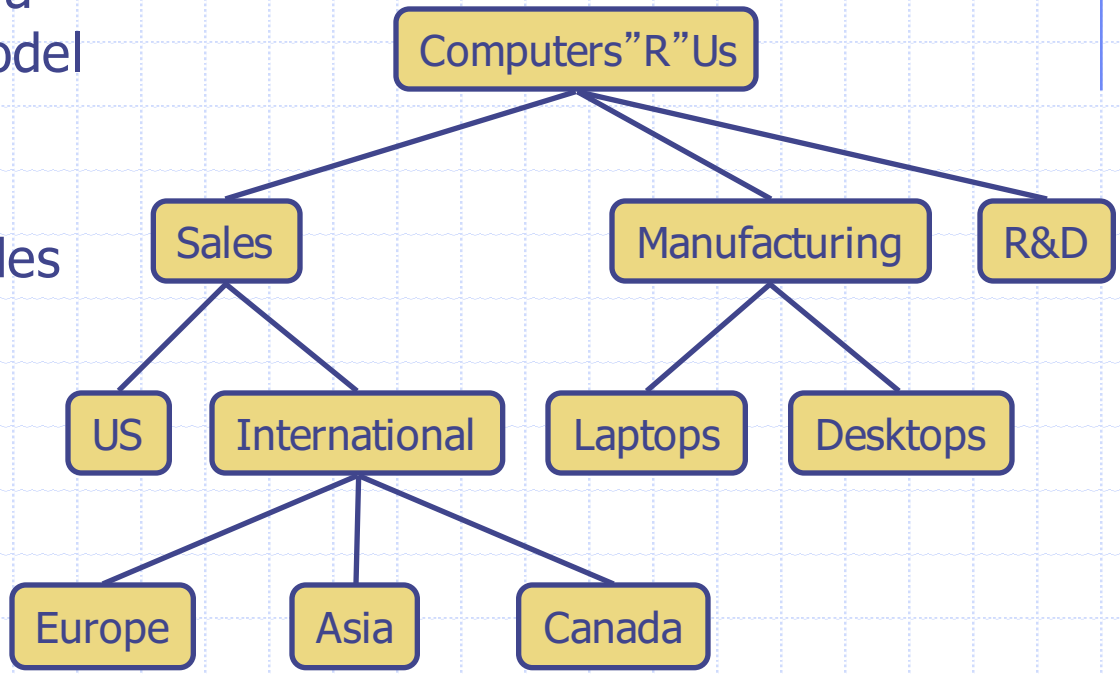
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Trees



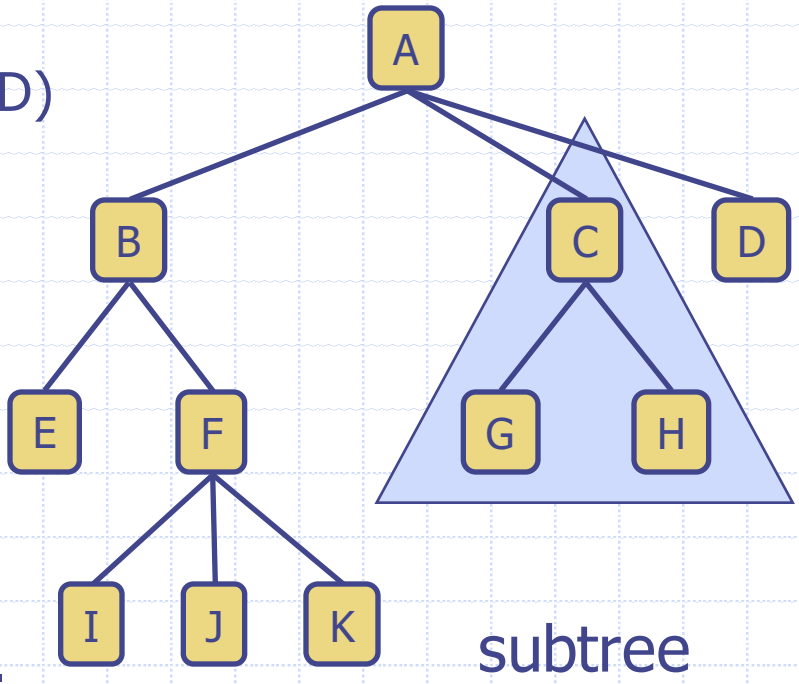
What is a Tree

- 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()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)

- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)

- Additional update methods may be defined by data structures implementing the Tree ADT

Java Interface

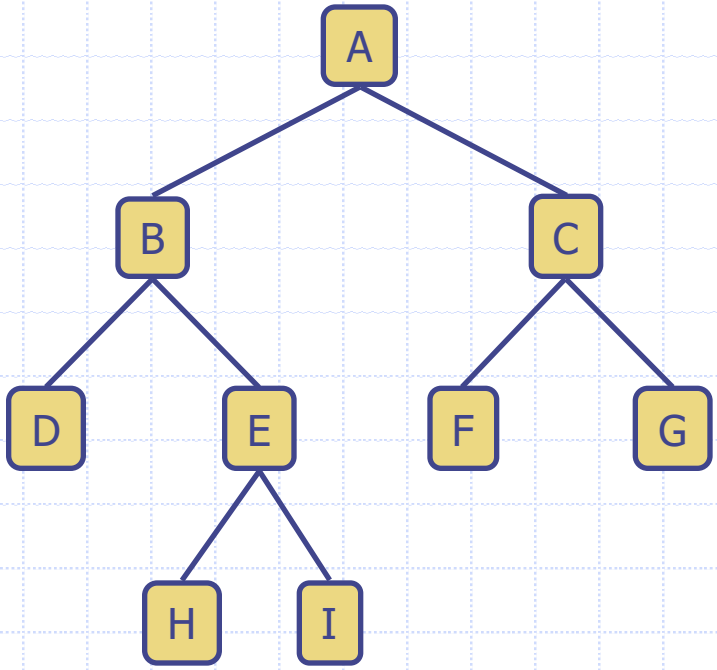
Methods for a Tree interface:

```
1  /** An interface for a tree where nodes can have an arbitrary number of children. */
2  public interface Tree<E> extends Iterable<E> {
3    Position<E> root( );
4    Position<E> parent(Position<E> p) throws IllegalArgumentException;
5    Iterable<Position<E>> children(Position<E> p)
6                                          throws IllegalArgumentException;
7    int numChildren(Position<E> p) throws IllegalArgumentException;
8    boolean isInternal(Position<E> p) throws IllegalArgumentException;
9    boolean isExternal(Position<E> p) throws IllegalArgumentException;
10   boolean isRoot(Position<E> p) throws IllegalArgumentException;
11   int size( );
12   boolean isEmpty( );
13   Iterator<E> iterator( );
14   Iterable<Position<E>> positions( );
15 }
```

Binary Trees

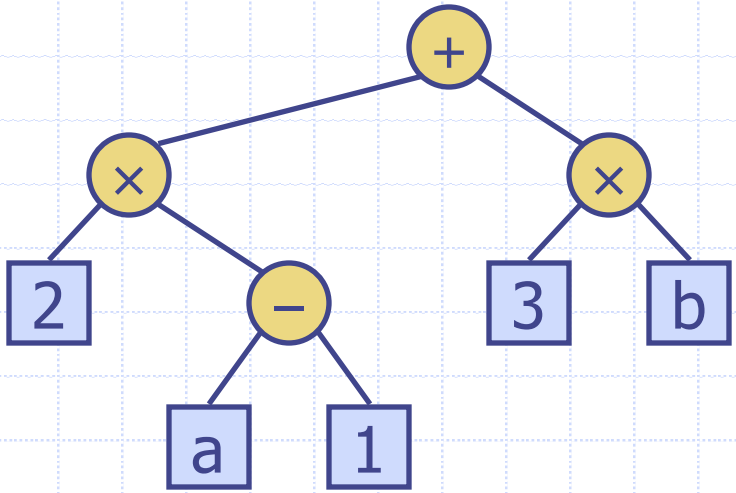
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary 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



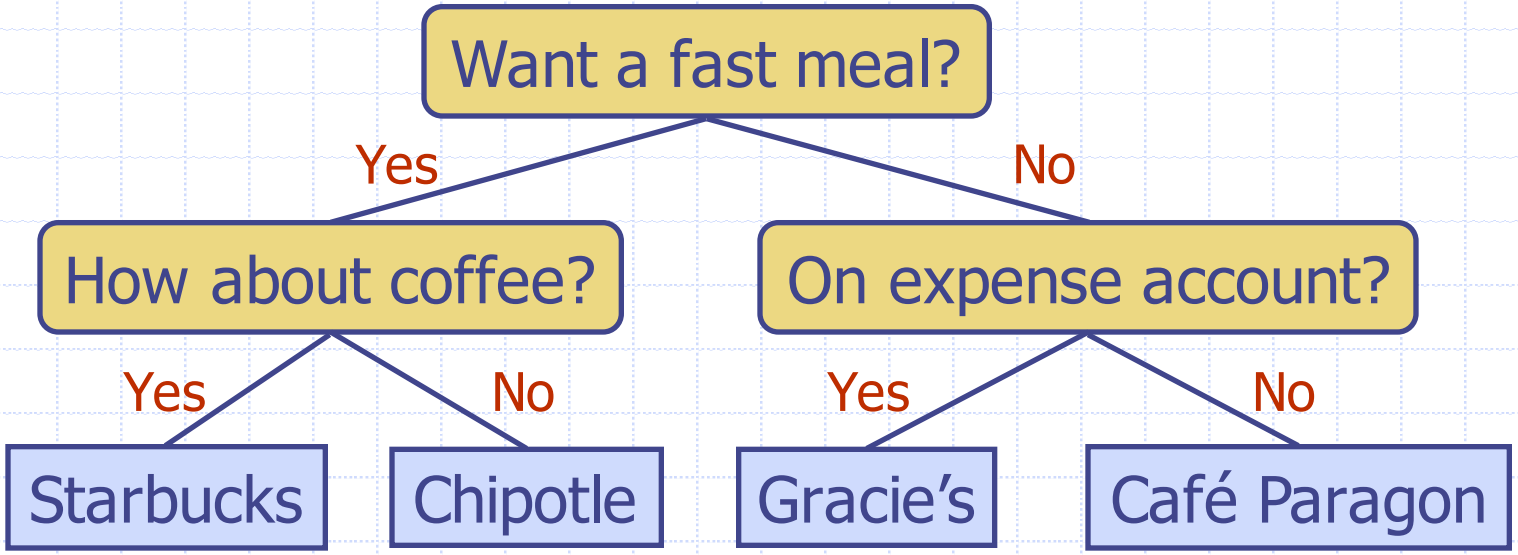
Arithmetic Expression Tree

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



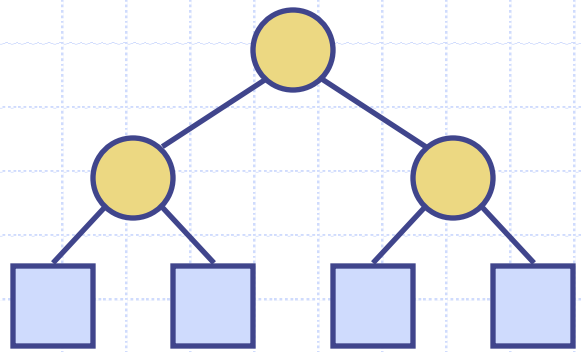
Decision Tree

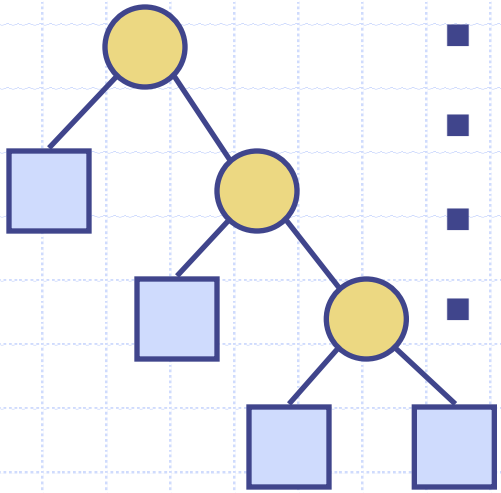
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Properties of Proper Binary Trees

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





- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

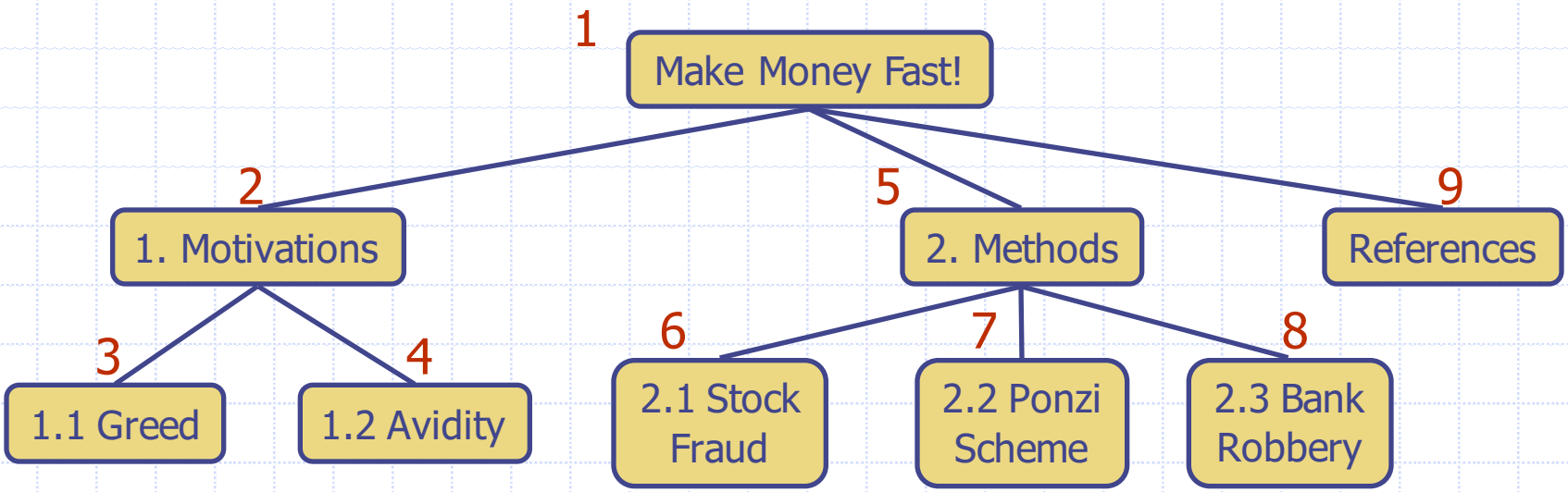
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - position sibling(p)
- The above methods return null when there is no left, right, or sibling of p, respectively
- Update methods may be defined by data structures implementing the BinaryTree 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

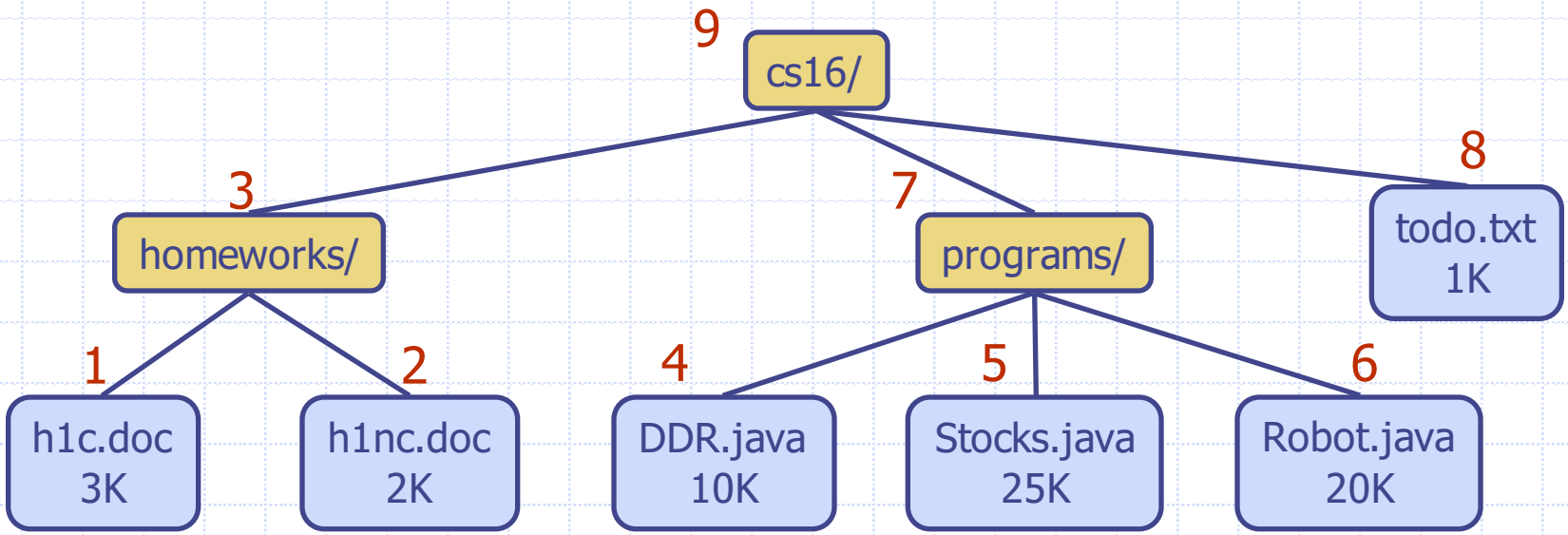
Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)



Postorder Traversal

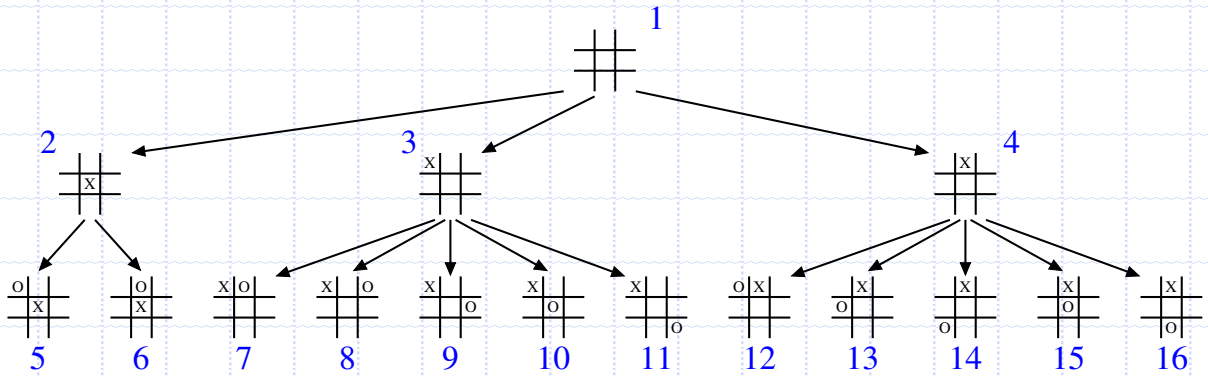
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

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



Breadth-First Traversal

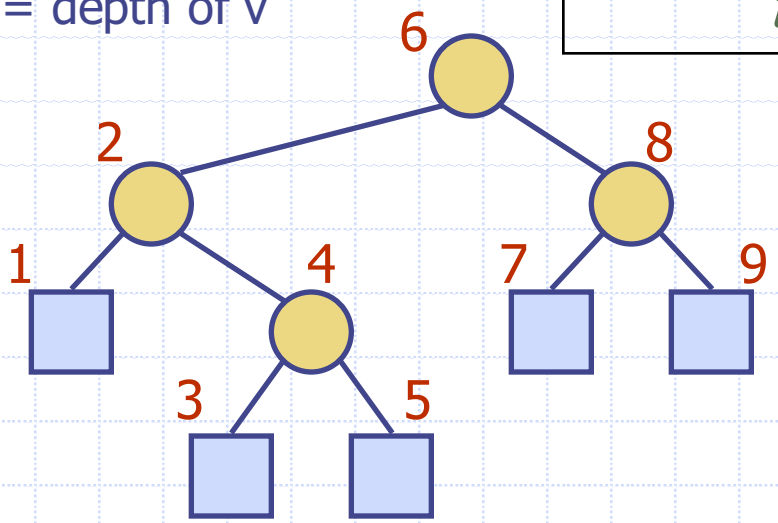
- Visit all positions at depth d before we visit the positions at depth d+1



Inorder Traversal

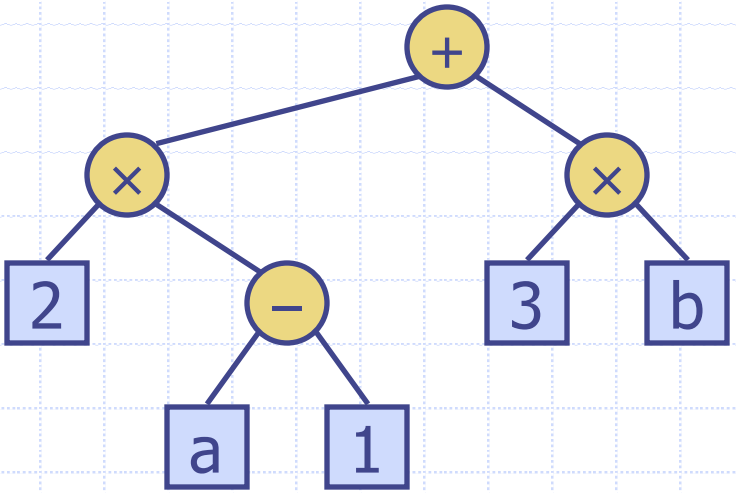
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
  if *left* (*v*) ≠ **null**
    inOrder (*left* (*v*))
  visit(*v*)
  if *right*(*v*) ≠ **null**
    inOrder (*right* (*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)

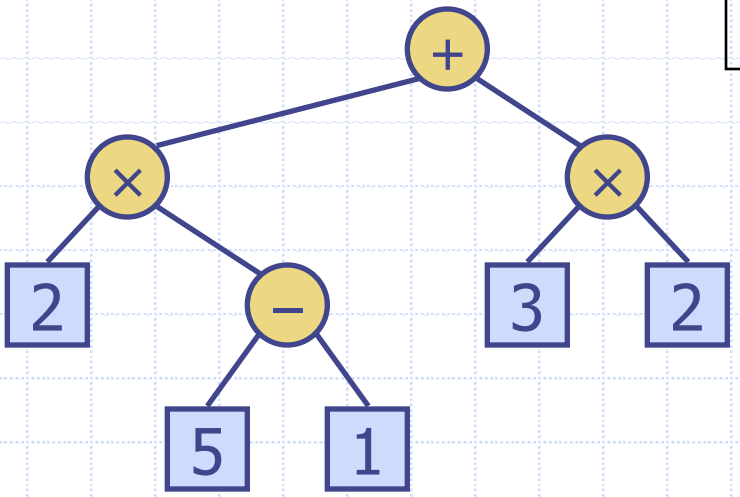
```
if left(v) ≠ null
    print("(")
    inOrder(left(v))
print(v.element())
if right(v) ≠ null
    inOrder(right(v))
    print(")")
```

((2 × (a − 1)) + (3 × b))

Evaluate Arithmetic Expressions

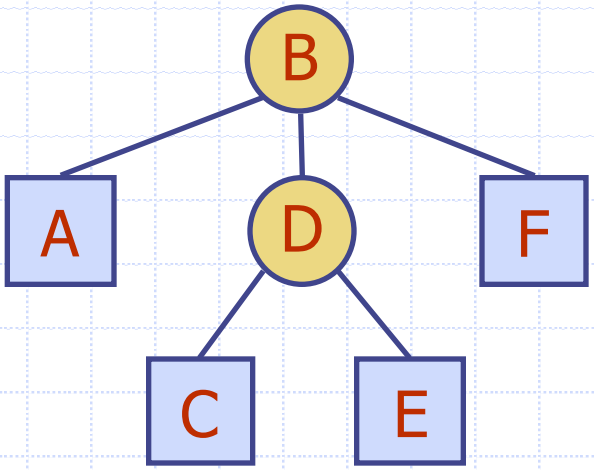
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

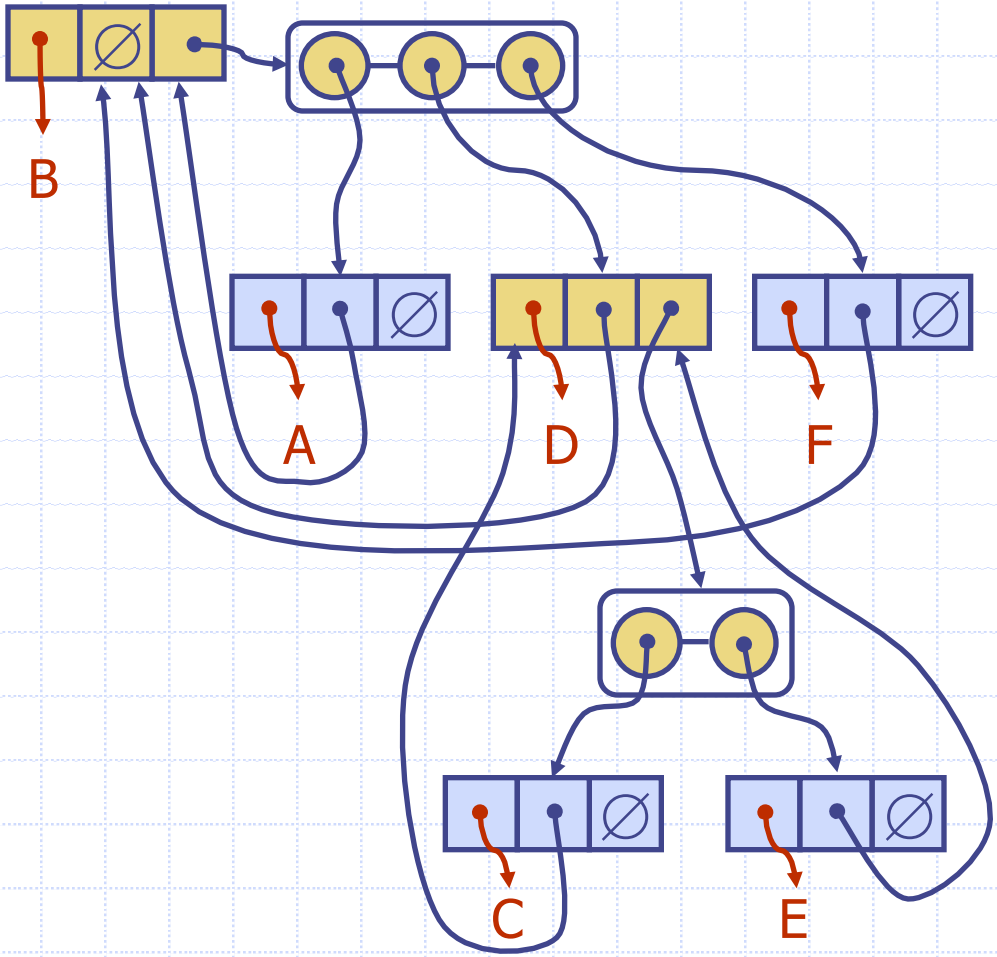
Algorithm *evalExpr*(*v*)
  if *isExternal* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*left*(*v*))
    $y \leftarrow$ *evalExpr*(*right*(*v*))
    $\lozenge \leftarrow$ operator stored at *v*
    return $x \lozenge y$



Linked Structure for Trees

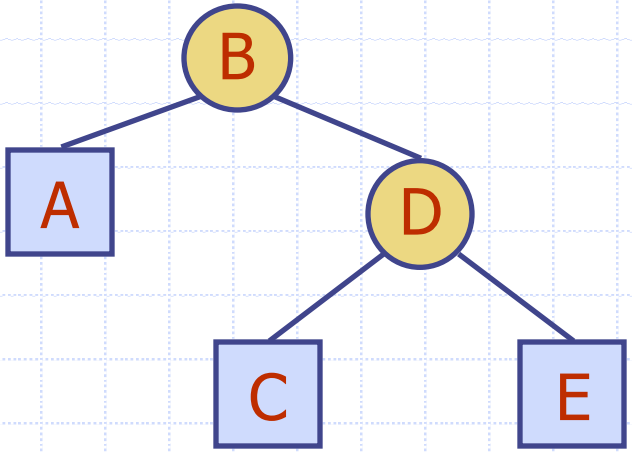
- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT

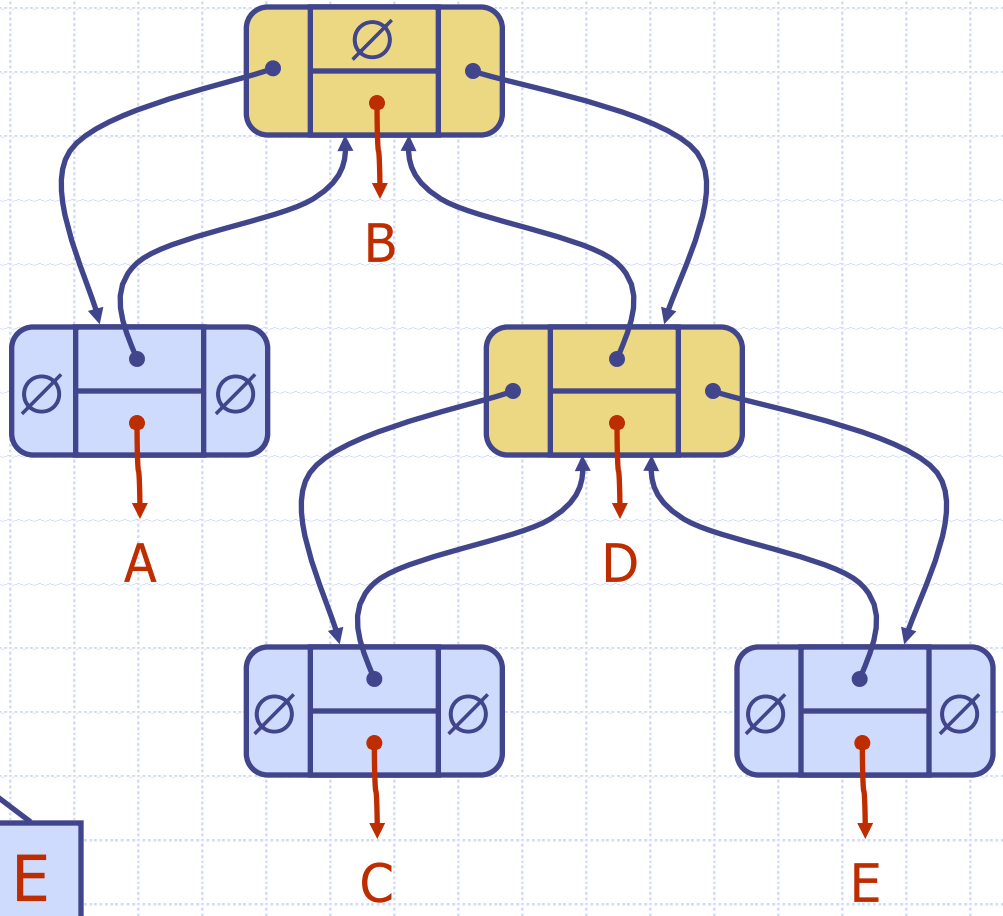




Linked Structure for Binary Trees

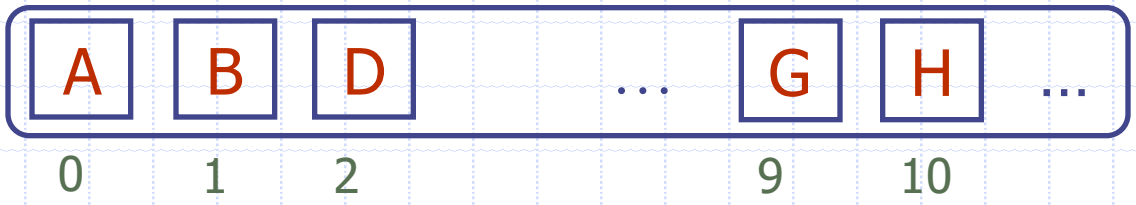
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT

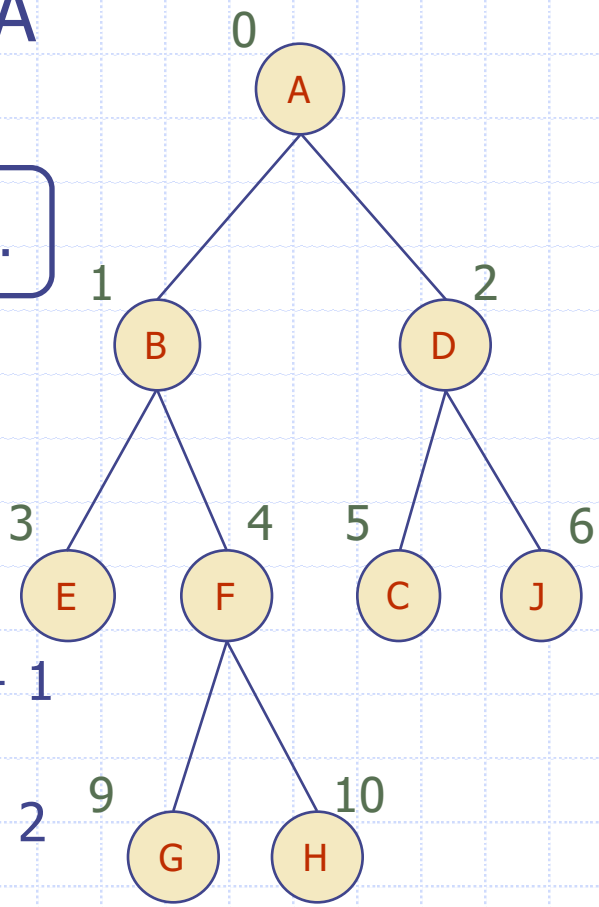




Array-Based Representation of Binary Trees

- Nodes are stored in an array A





Node v is stored at A[rank(v)]

- rank(root) = 0
- if node is the left child of parent(node),
  rank(node) = 2 · rank(parent(node)) + 1
- if node is the right child of parent(node),
  rank(node) = 2 · rank(parent(node)) + 2