APA 254
Data Structures

Lecture 8.1

(Binary Trees)

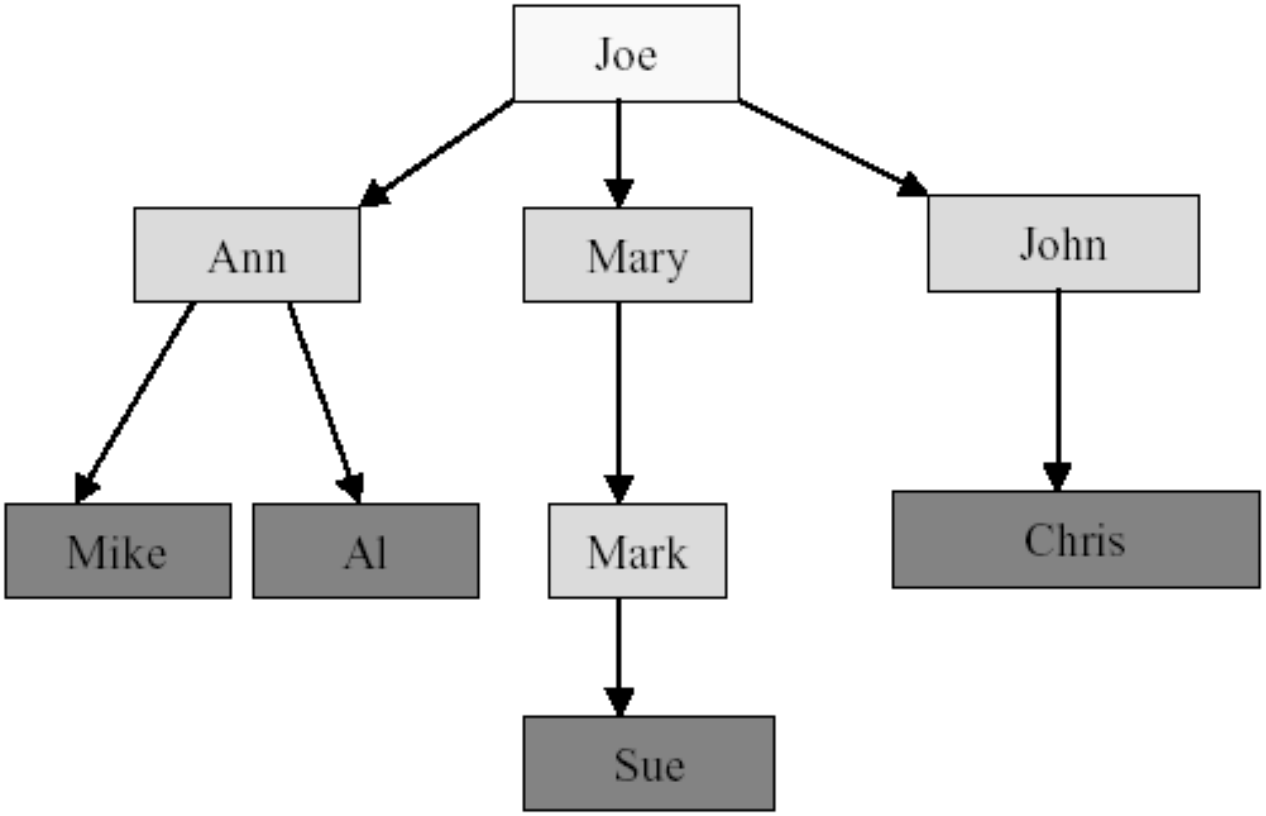
Dept. of Information System

Hanyang University

Linear Lists and Trees

- Linear lists are useful for serially ordered data
  - $(e_1, e_2, e_3, \ldots, e_n)$
  - Days of week
  - Months in a year
  - Students in a class
- Trees are useful for hierarchically ordered data
  - Joe's descendants
  - Corporate structure
  - Government subdivisions
  - Software structure
- Read Examples 11.1-11.4

Joe's Descendants

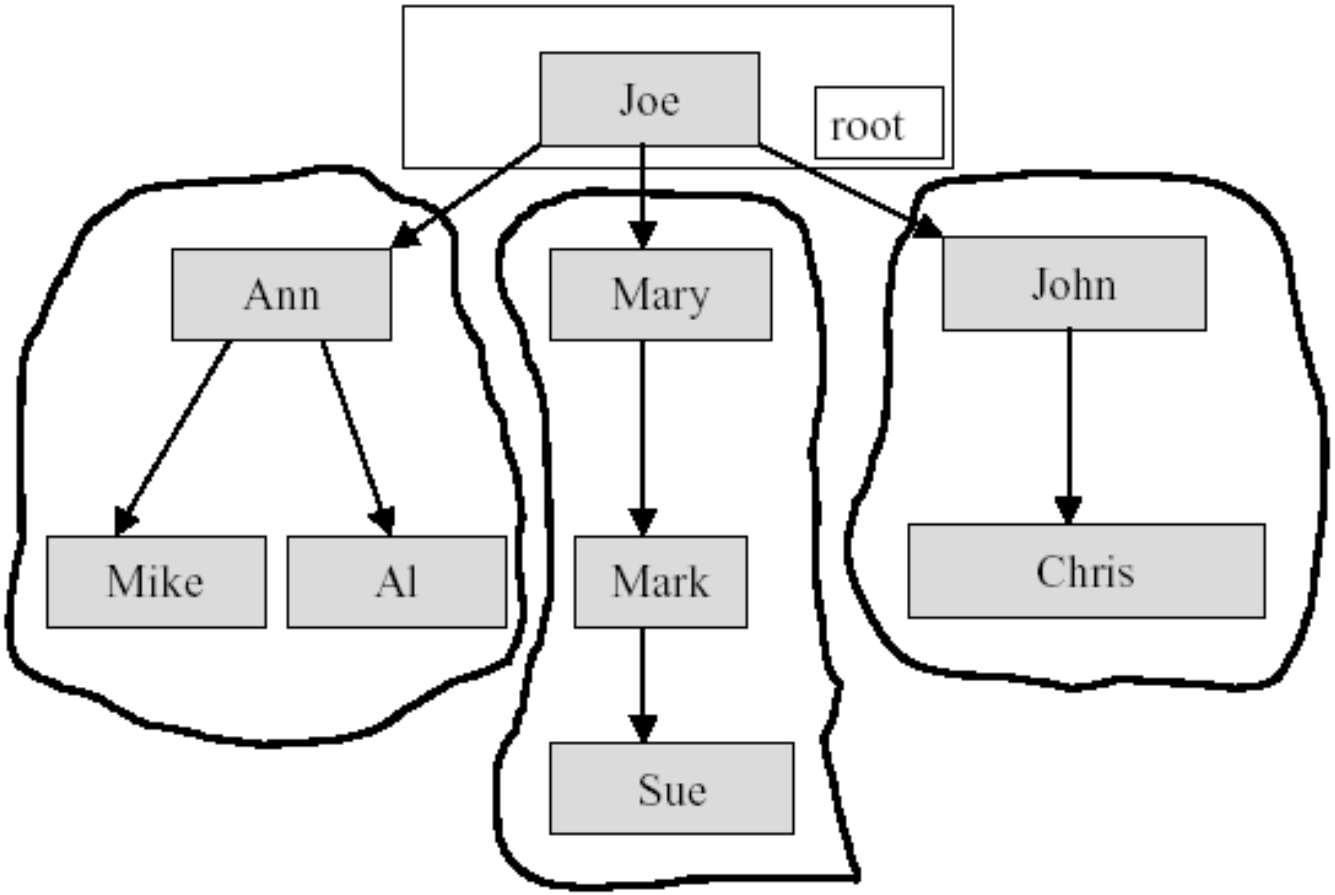


What are other examples of hierarchically ordered data?

Definition of Tree

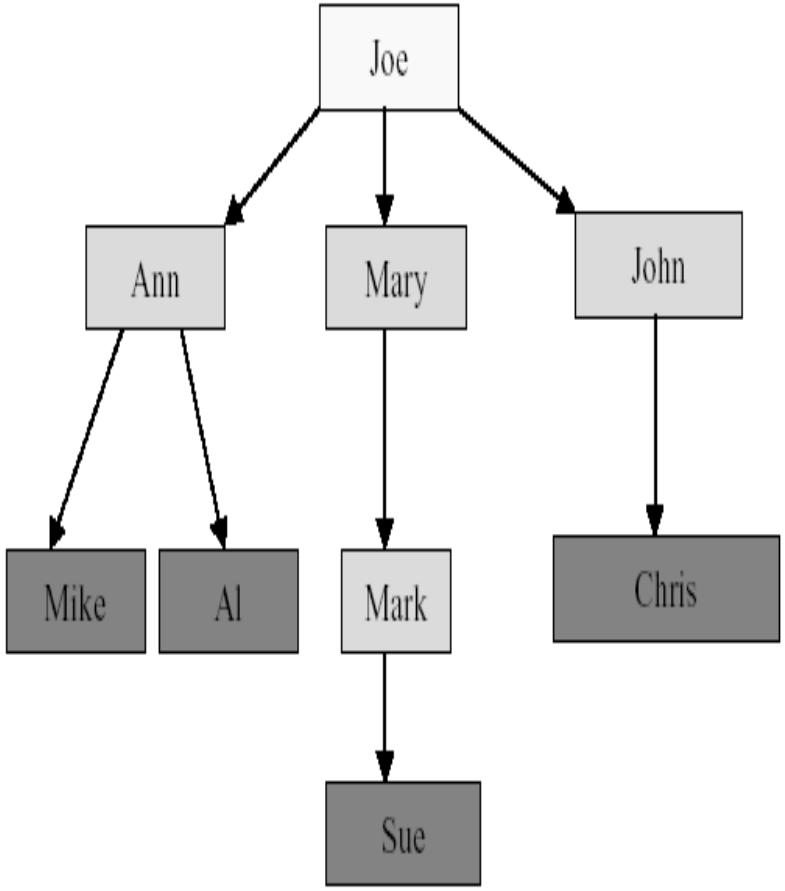
- A tree *t* is a finite nonempty set of elements
- One of these elements is called the root
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of *t*.

Subtrees



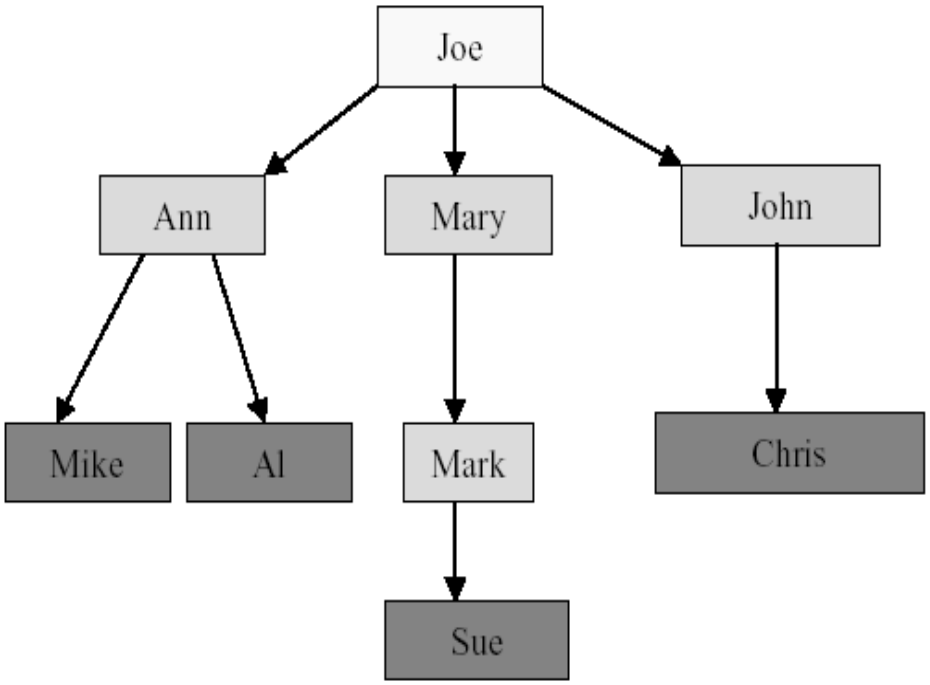
Tree Terminology

- The element at the top of the hierarchy is the **root**.
- Elements next in the hierarchy are the **children** of the root.
- Elements next in the hierarchy are the **grandchildren** of the root, and so on.
- Elements at the lowest level of the hierarchy are the **leaves**.



Other Definitions

- Leaves, Parent, Grandparent, Siblings, Ancestors, Descendents



Leaves = {Mike,Al,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

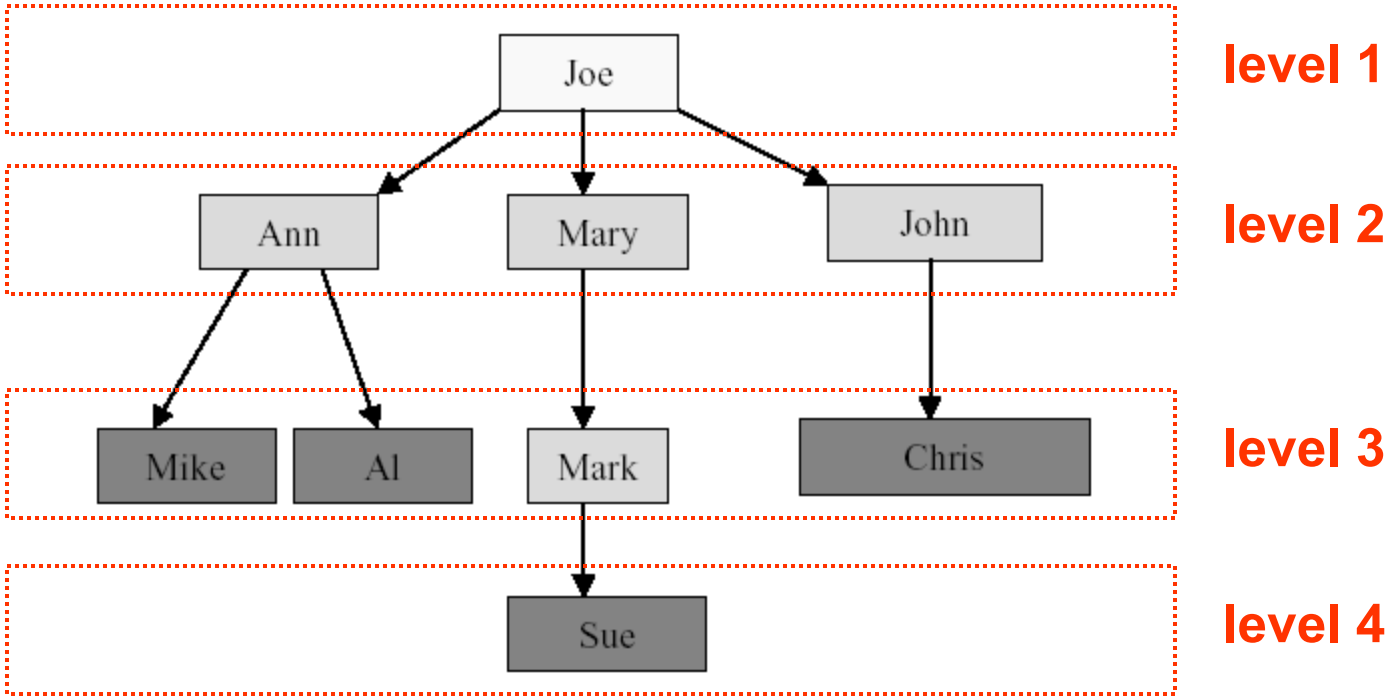
Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}

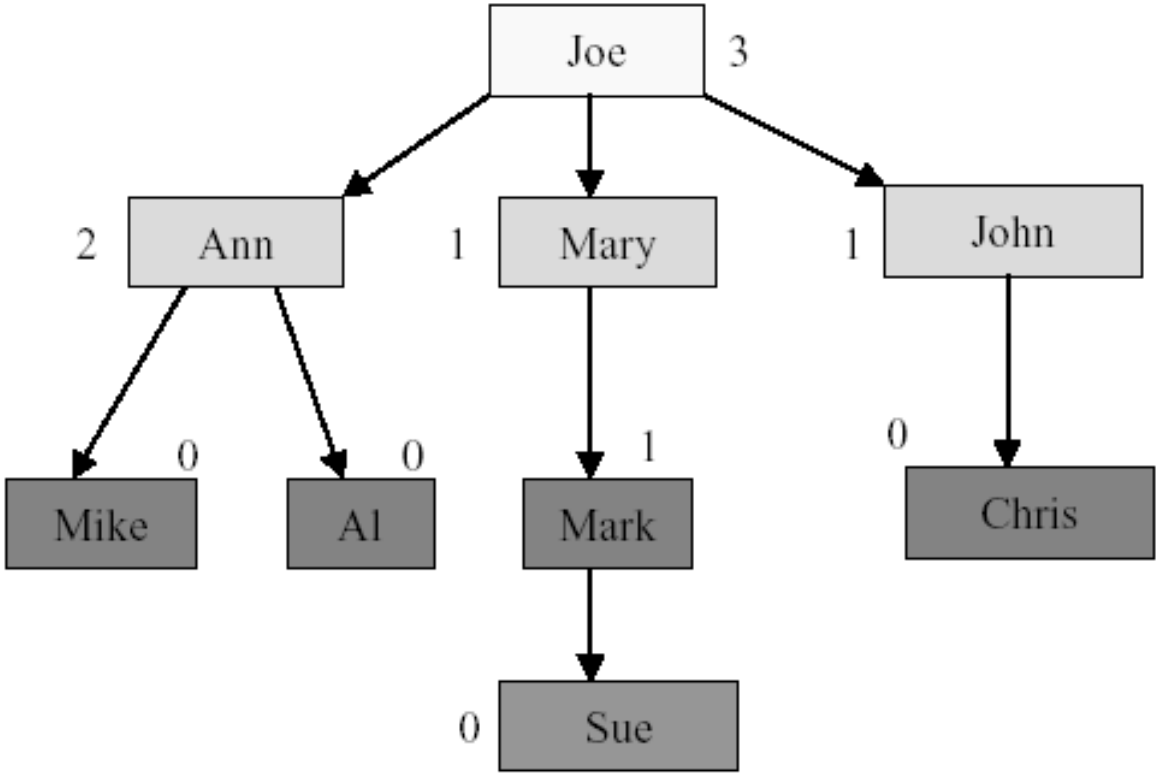
Levels and Height

- Root is at level 1 and its children are at level 2.
- Height = depth = number of levels



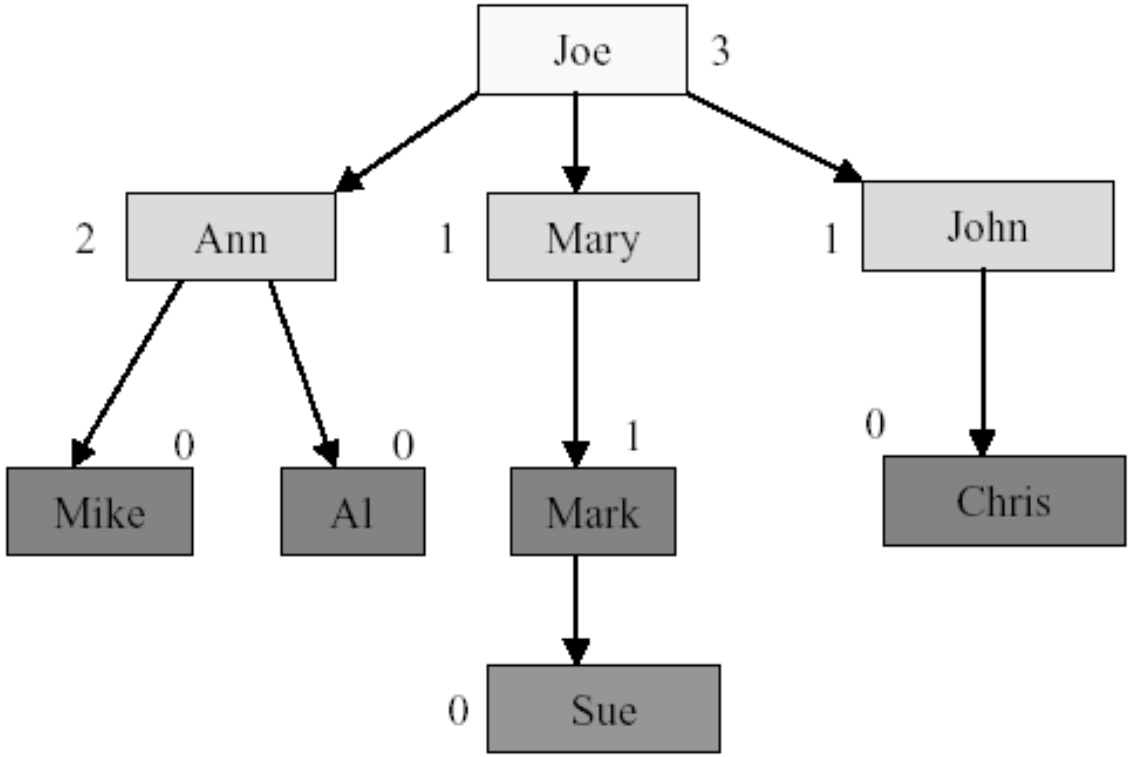
Node Degree

- Node degree is the number of children it has



Tree Degree

- Tree degree is the maximum of node degrees



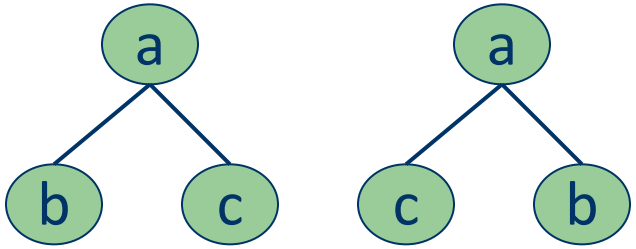
tree degree = 3

Binary Tree

- A nonempty binary tree has a root element and the remaining elements (if any) are partitioned into two binary trees
- They are called the left and right subtrees of the binary tree

Difference Between a Tree & a Binary Tree

- A binary tree may be empty; a tree cannot be empty.
- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- The subtrees of a binary tree are ordered; those of a tree are not ordered.



- different when viewed as a binary tree

- same when viewed as a tree

Binary Tree for Expressions

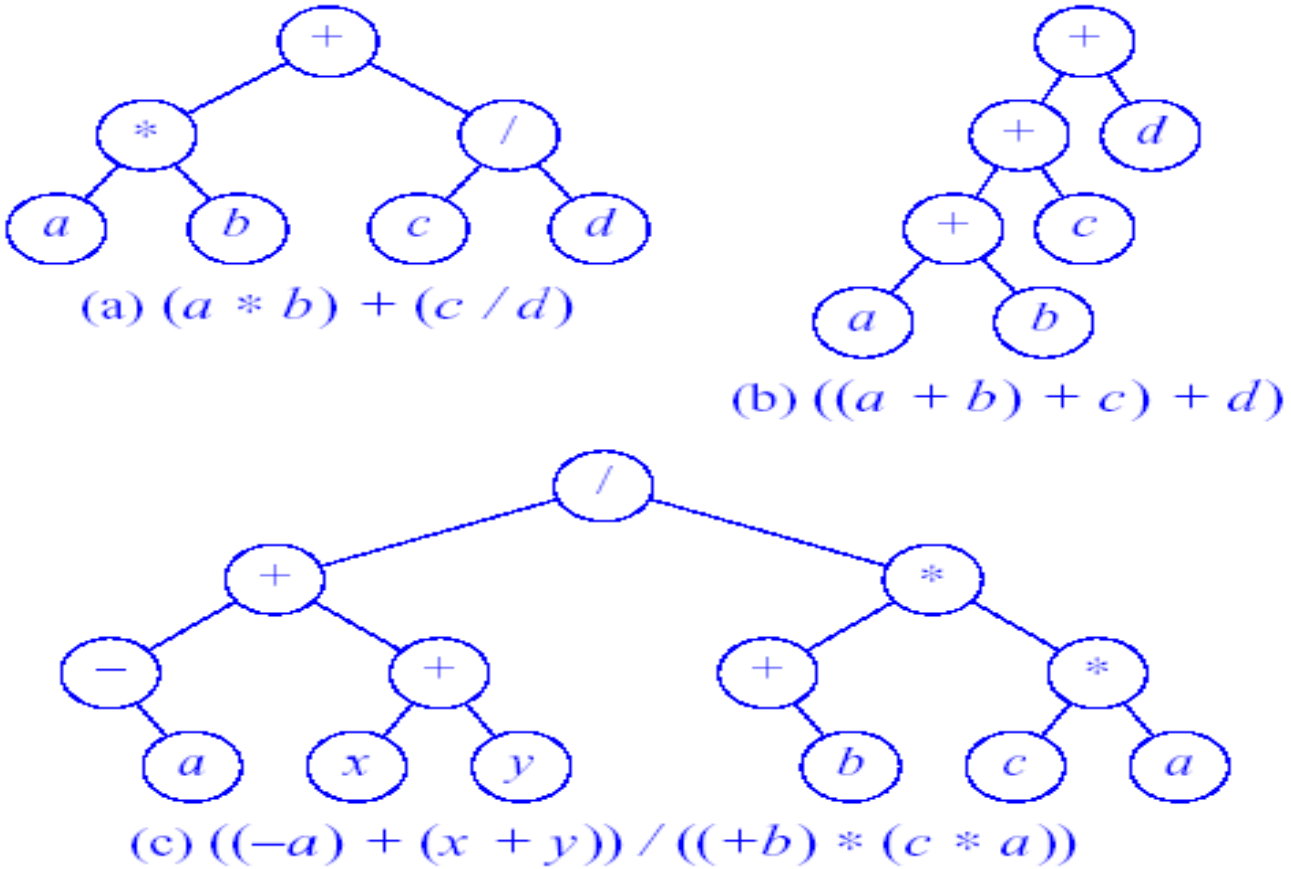


Figure 11.5 Expression Trees

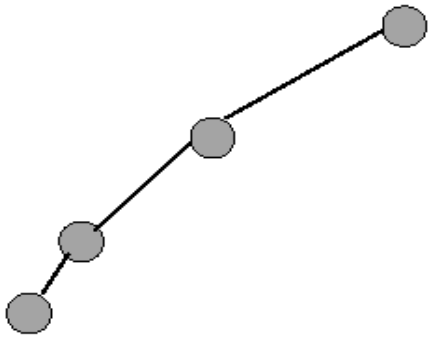
Binary Tree Properties

1. The drawing of every binary tree with n elements, $n > 0$, has exactly n-1 edges. (see its proof on pg. 426)

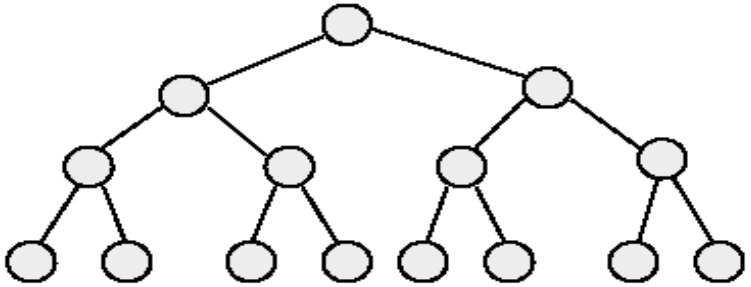
2. A binary tree of height h, $h >= 0$, has at least h and at most 2^h-1 elements in it. (see its proof on pg. 427)

Binary Tree Properties

3. The height of a binary tree that contains n elements, $n >= 0$, is at least $\lceil (log_2(n+1)) \rceil$ and at most n. (see its proof on pg. 427)



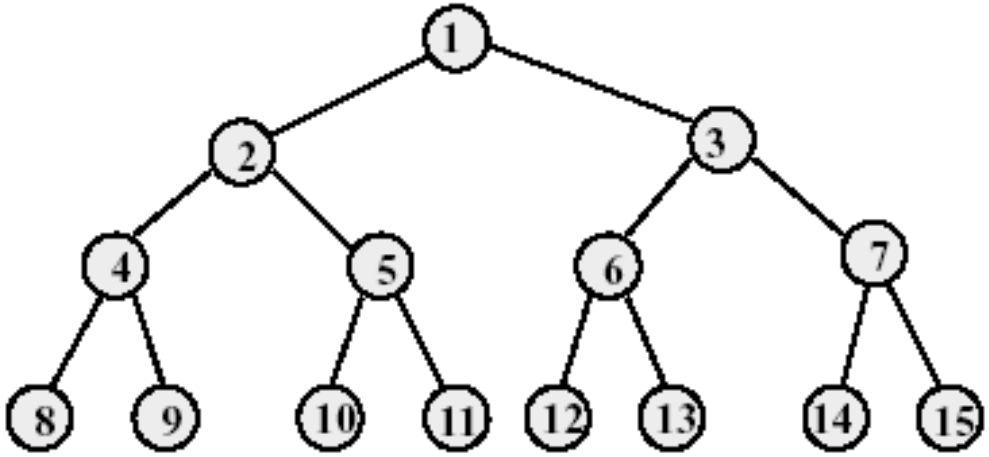
minimum number of elements



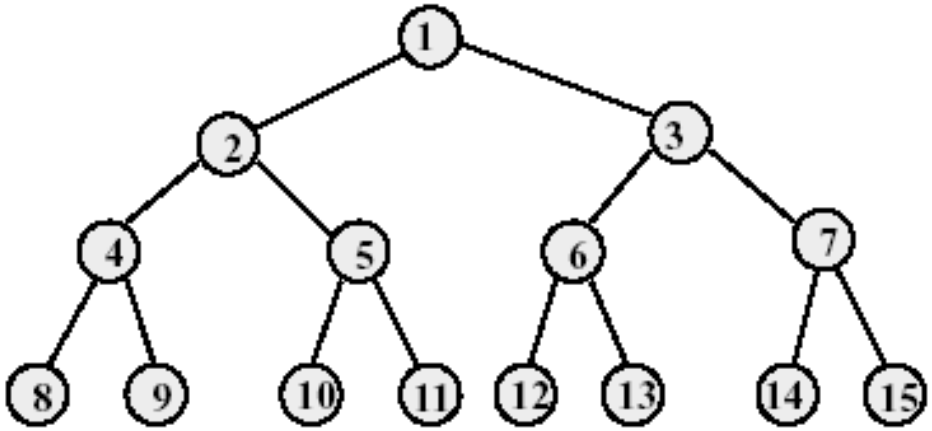
maximum number of elements

Full Binary Tree

- A full binary tree of height h has exactly 2^h-1 nodes.
- Numbering the nodes in a full binary tree
  - Number the nodes 1 through 2^h-1
  - Number by levels from top to bottom
  - Within a level, number from left to right (see Fig. 11.6)

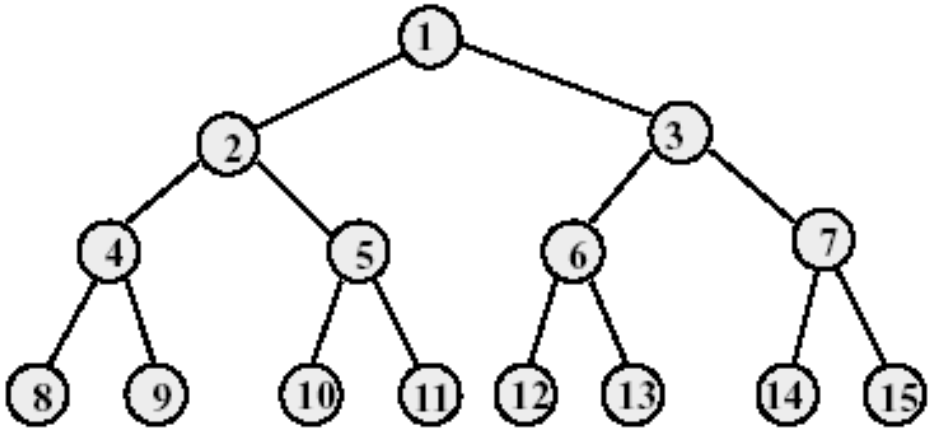


Node Number Property of Full Binary Tree



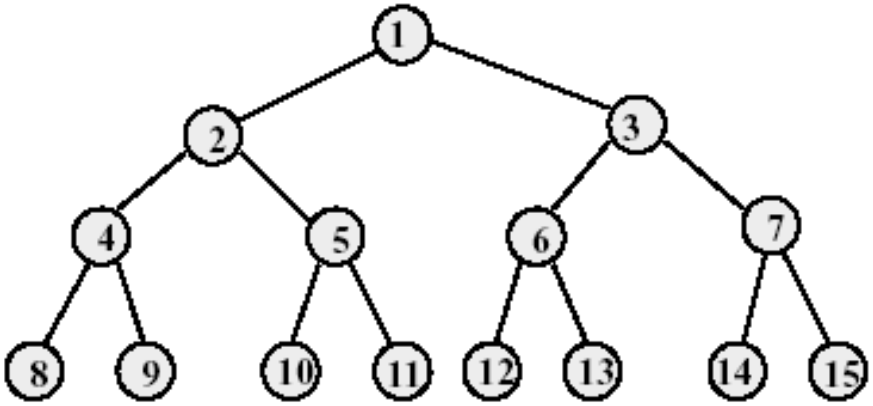
- Parent of node *i* is node $\lfloor (i/2) \rfloor$, unless i = 1
- Node 1 is the root and has no parent

Node Number Property of Full Binary Tree



- Left child of node *i* is node *2i*, unless *2i* > *n*, where *n* is the total number of nodes.
- If *2i* > *n*, node *i* has no left child.

Node Number Property of Full Binary Tree



- Right child of node *i* is node *2i*+1, unless *2i*+1 > *n*, where n is the total number of nodes.
- If *2i+1* > *n*, node *i* has no right child.

Complete Binary Tree with N Nodes

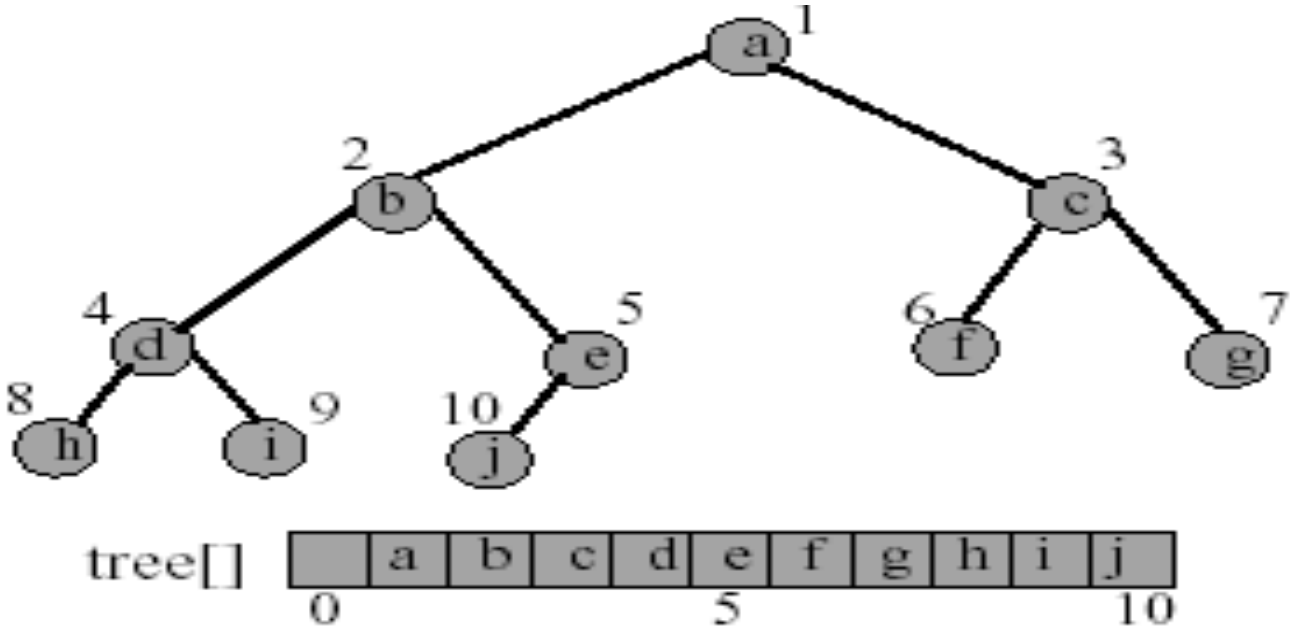
- Start with a full binary tree that has at least *n* nodes
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through *n* is the *n*-node complete binary tree.
- A full binary tree is a special case of a complete binary tree
- See Figure 11.7 for complete binary tree examples
- See Figure 11.8 for incomplete binary tree examples

Binary Tree Representation

- Array representation
- Linked representation

Array Representation of Binary Tree

- The binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.



Incomplete Binary Trees

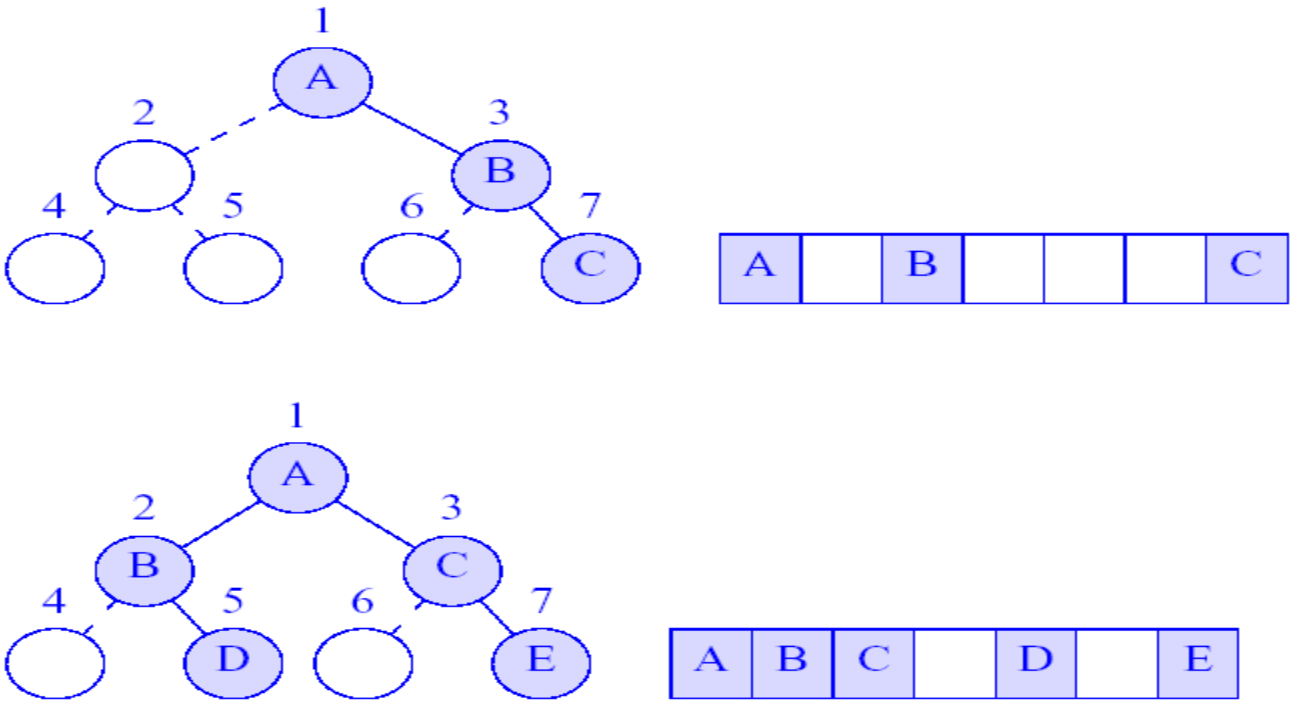
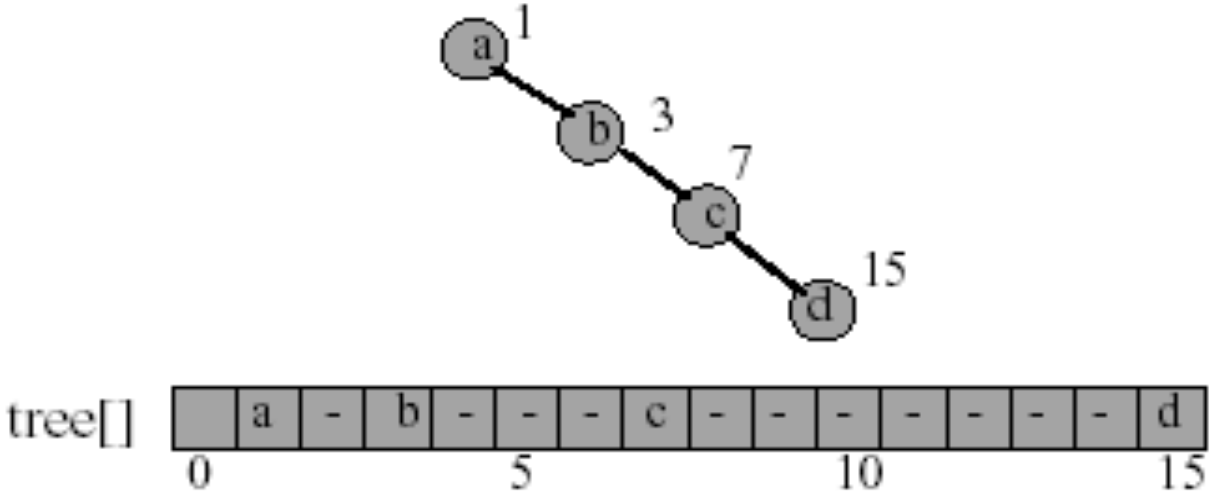


Fig. 11.8 Incomplete binary trees

- Complete binary tree with some missing elements

Right-Skewed Binary Tree

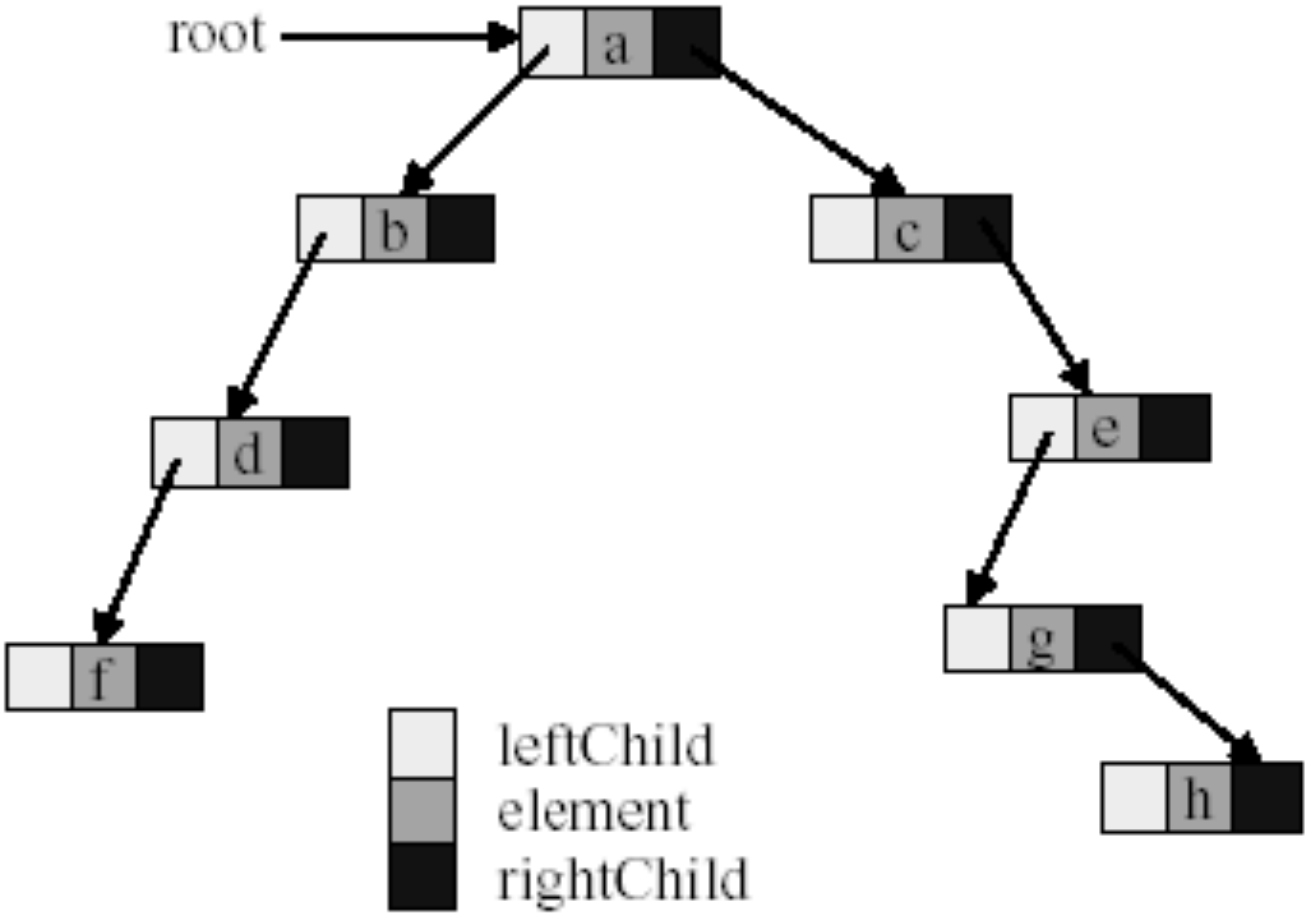


- An *n* node binary tree needs an array whose length is between ***n+1*** and 2^n.
- Right-skewed binary tree wastes the most space
- What about left-skewed binary tree?

Linked Representation of Binary Tree

- The most popular way to present a binary tree
- Each element is represented by a node that has two link fields (leftChild and rightChild) plus an element field (see Figure 11.10)
- Each binary tree node is represented as an object whose data type is binaryTreeNode (see Program 11.1)
- The space required by an *n* node binary tree is *n* * sizeof(binaryTreeNode)

Linked Representation of Binary Tree



Node Class For Linked Binary Tree

```
template<class T>
class binaryTreeNode {
private:
    T element;
    binaryTreeNode<T> *leftChild, *rightChild;
public:                                                  // 3 constructors
    binaryTreeNode() { leftChild = rightChild = NULL; } // no params
    binaryTreeNode(const T& theElement) {     // element param only
          element = theElement; leftChild = rightChild = NULL;
    }
    binaryTreeNode(const T& theElement, binaryTreeNode *l,
                   binaryTreeNode *r) {    // element + links params
          element = theElement; leftChild = l; rightChild = r;
    }
}
```

Common Binary Tree Operations

- Determine the height
- Determine the number of nodes
- Make a copy
- Determine if two binary trees are identical
- Display the binary tree
- Delete a tree
- If it is an expression tree, evaluate the expression
- If it is an expression tree, obtain the parenthesized form of the expression

Binary Tree Traversal

- Many binary tree operations are done by performing a **traversal** of the binary tree
- In a traversal, each element of the binary tree is **visited** exactly once
- During the visit of an element, all actions (make a copy, display, evaluate the operator, etc.) with respect to this element are taken

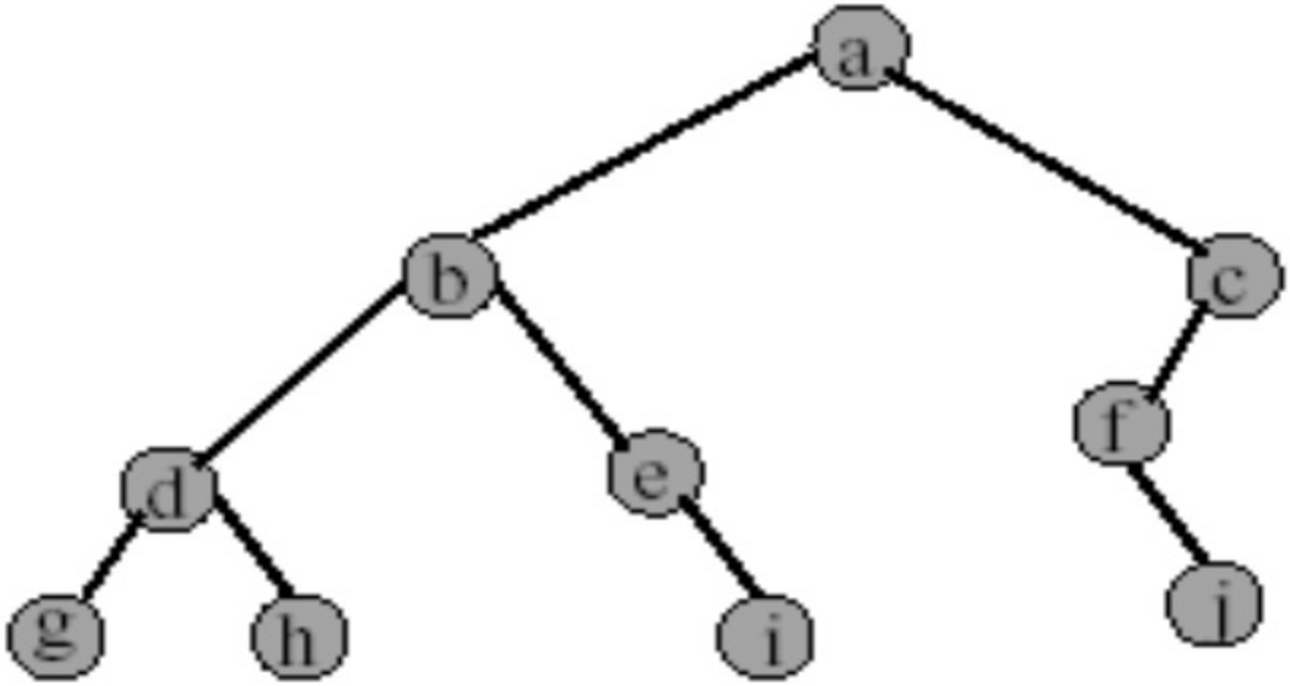
Binary Tree Traversal Methods

- **Preorder**
  - The root of the subtree is processed first before going into the left then right subtree (root, left, right).
- **Inorder**
  - After the complete processing of the left subtree the root is processed followed by the processing of the complete right subtree (left, root, right).
- **Postorder**
  - The root is processed only after the complete processing of the left and right subtree (left, right, root).
- **Level order**
  - The tree is processed by levels. So first all nodes on level i are processed from left to right before the first node of level i+1 is visited

Preorder Traversal

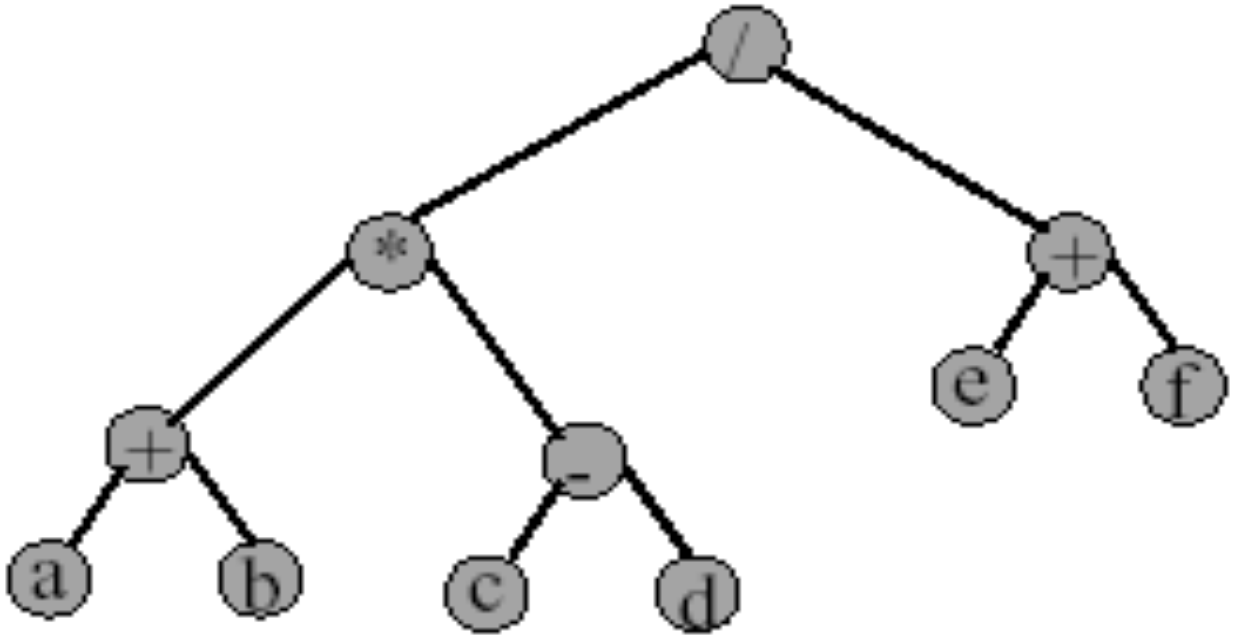
```
template<class T>   // Program 11.2
void preOrder(binaryTreeNode<T> *t)
{
   if (t != NULL) {
         visit(t);                          // visit tree root
         preOrder(t->leftChild);            // do left subtree
         preOrder(t->rightChild);           // do right subtree
   }
}
```

Preorder Example (visit = print)



a b d g h e i c f j

Preorder of Expression Tree



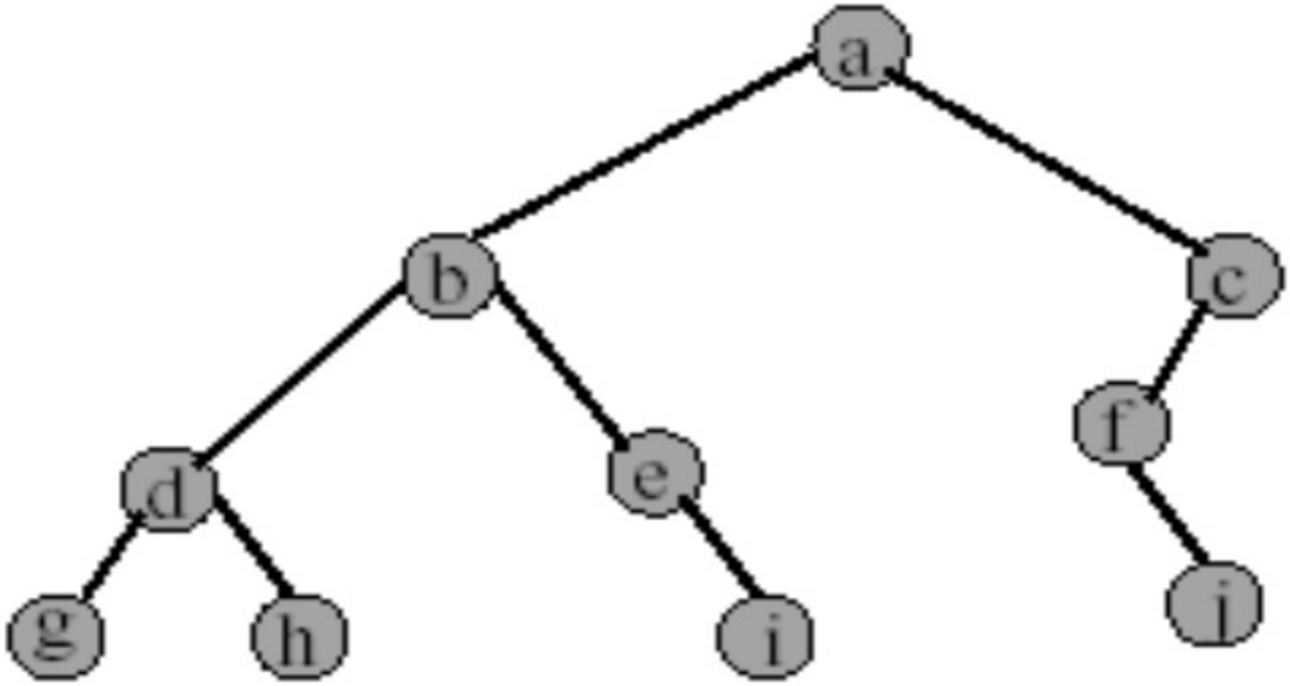
/ * + a b - c d + e f

Gives prefix form of expression.

Inorder Traversal

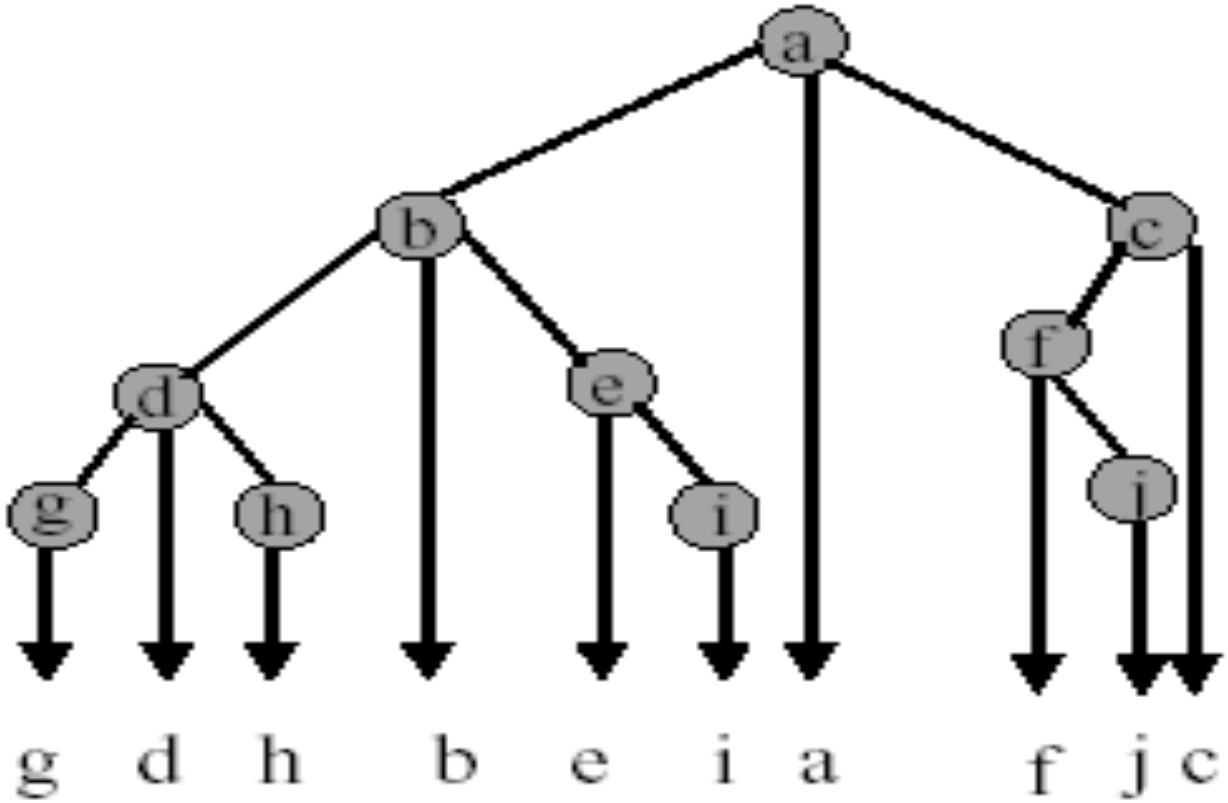
```
template<class T> // Program 11.3
void inOrder(binaryTreeNode<T> *t)
{
   if (t != NULL) {
         inOrder(t->leftChild);          // do left subtree
         visit(t);                       // visit tree root
         inOrder(t->rightChild);         // do right subtree
   }
}
```

Inorder Example (visit = print)

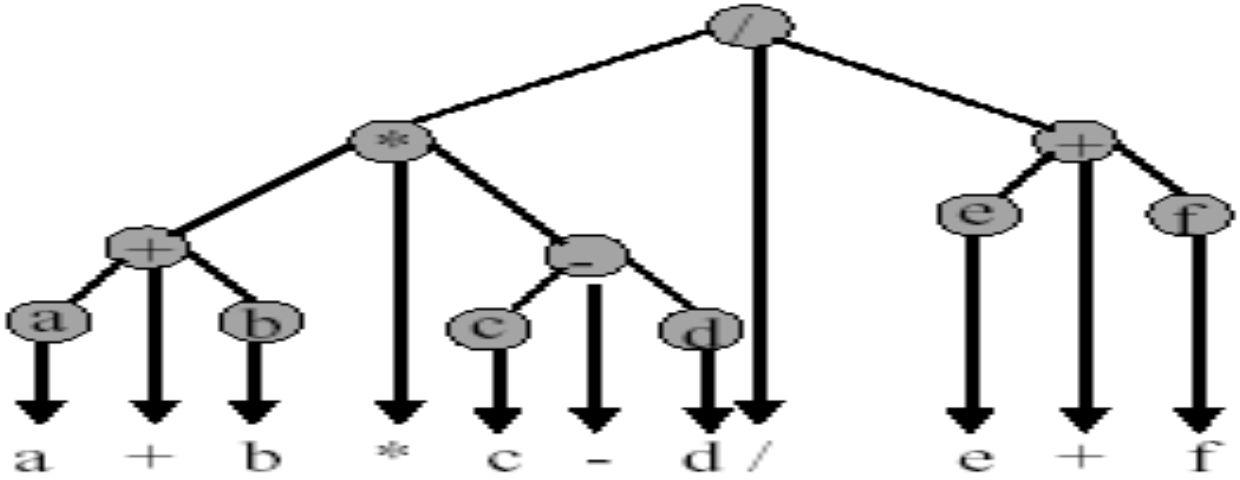


g d h b e i a f j c

Inorder by Projection



Inorder of Expression Tree

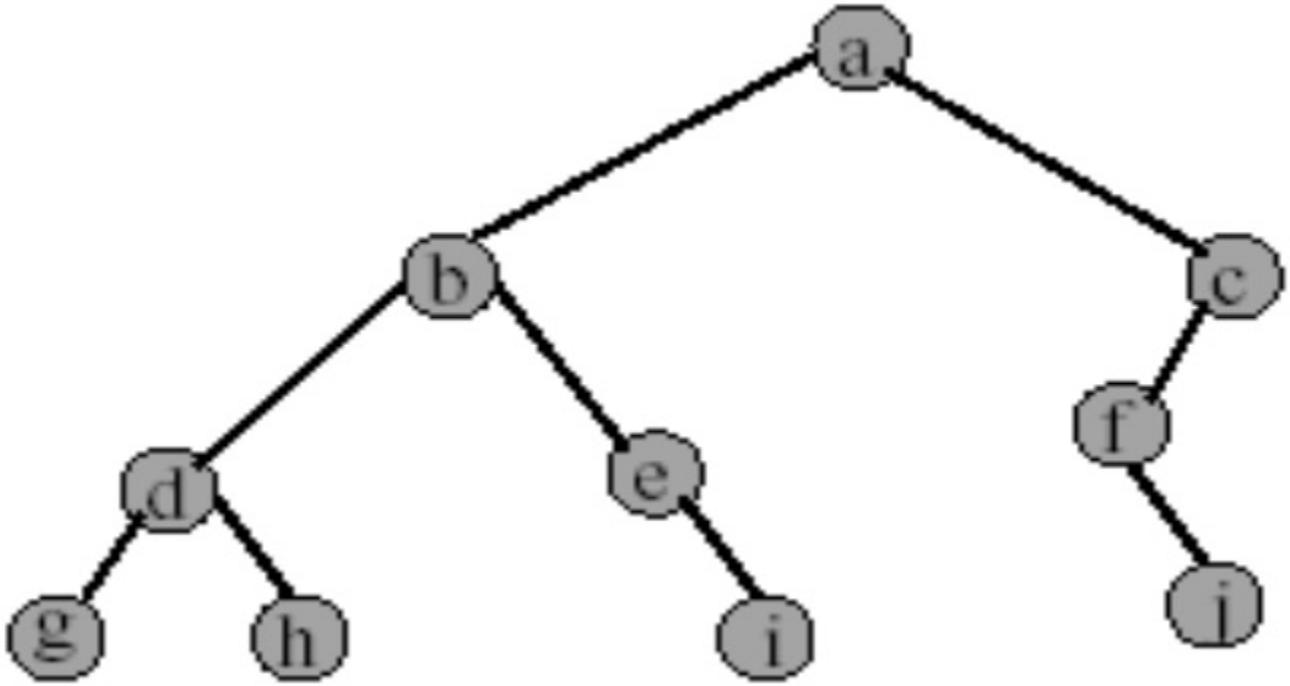


- Gives infix form of expression, which is how we normally write math expressions. What about parentheses?
- See Program 11.6 for parenthesized infix form
- Fully parenthesized output of the above tree?

Postorder Traversal

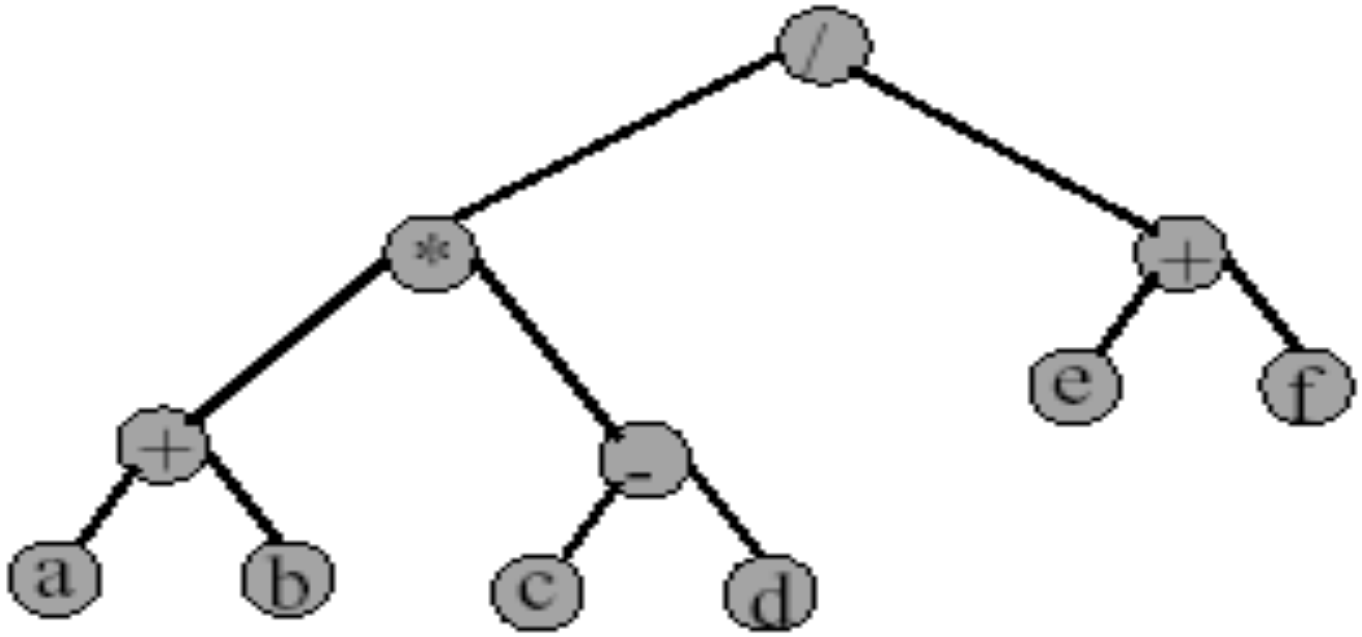
```
template<class T> // Program 11.4
void postOrder(binaryTreeNode<T> *t)
{
   if (t != NULL) {
         postOrder(t->leftChild);          // do left subtree
         postOrder(t->rightChild);         // do right subtree
         visit(t);                         // visit tree root
   }
}
```

Postorder Example (visit = print)



g h d i e b j f c a

Postorder of Expression Tree



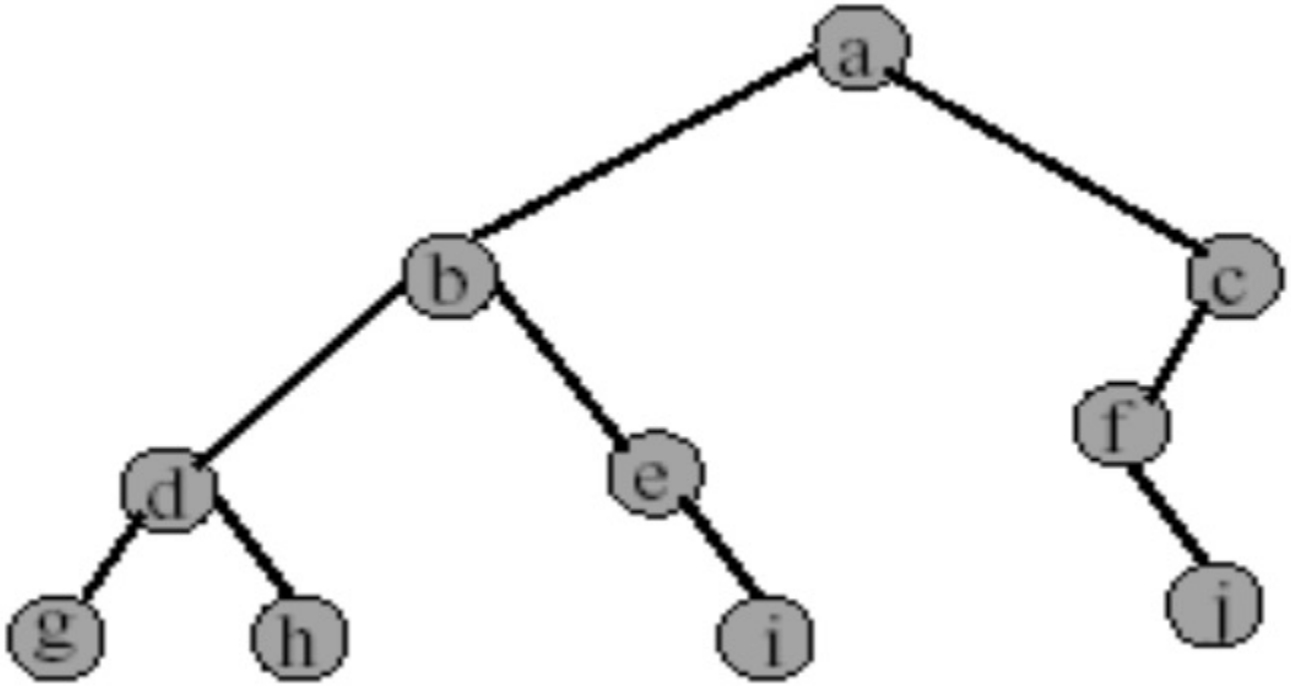
a b + c d - * e f + /

Gives postfix form of expression.

Level Order Traversal

```
template <class T> // Program 11.7
void levelOrder(binaryTreeNode<T> *t)
{// Level-order traversal of *t.
   linkedQueue<binaryTreeNode<T>*> Q;
   while (t != NULL) {
      visit(t);  // visit t
      if (t->leftChild) q.push(t->leftChild);        // put t's children
      if (t->rightChild) q.push(t->rightChild);   // on queue
      try {t = q.front();}                          // get next node to visit
      catch (queueEmpty) {return;}
      q.pop()
      }
 }
```

Level Order Example (visit = print)



- Add and delete nodes from a queue
- Output: a b c d e f g h i j

Space and Time Complexity

- The space complexity of each of the four traversal algorithm is **O(n)**
- The time complexity of each of the four traversal algorithm is **Θ(n)**

Binary Tree ADT, Class, Extensions

- See ADT 11.1 for Binary Tree ADT definition
- See Program 11.8 for Binary Tree abstract class
- See Programs 11.9 - 11.11 for operation implementations

- Read Sections 11.1 - 11.8