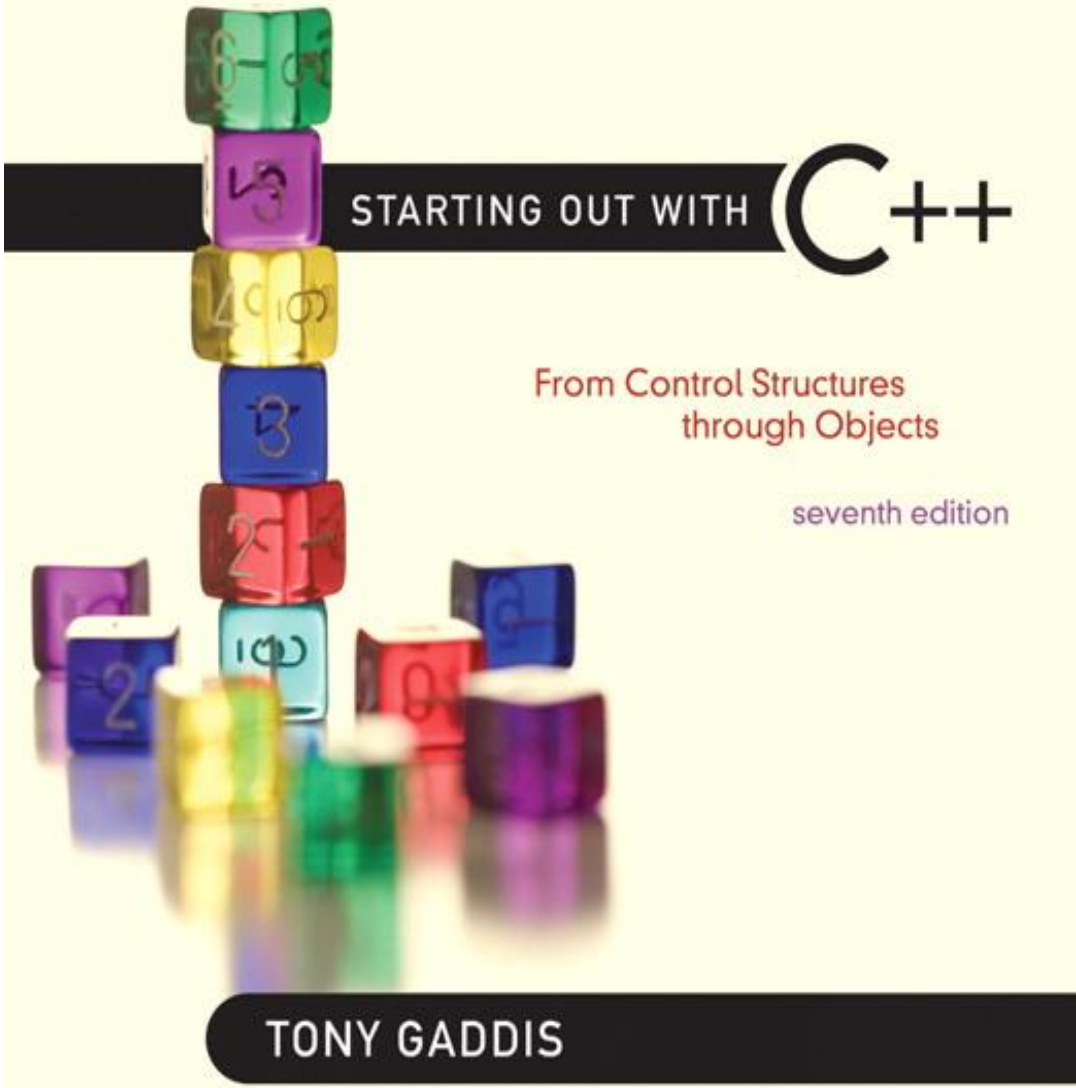
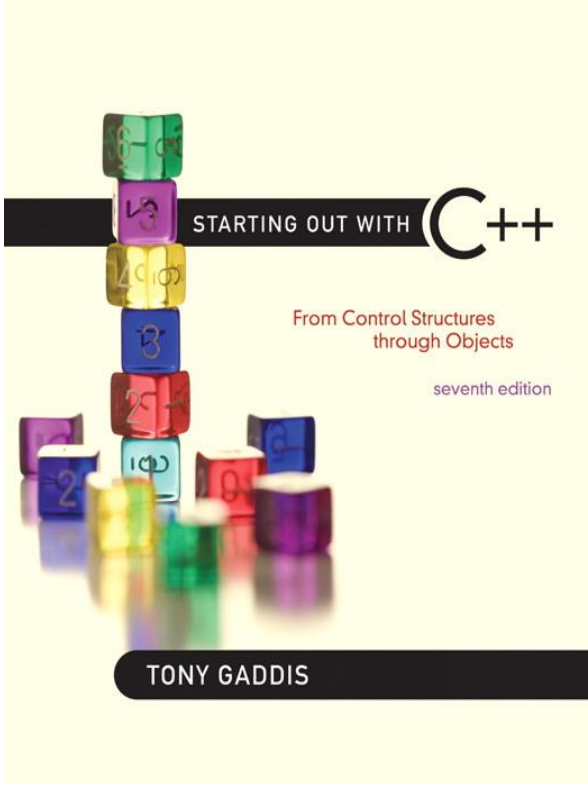
Chapter 20:

Binary Trees



Addison-Wesley
is an imprint of
PEARSON



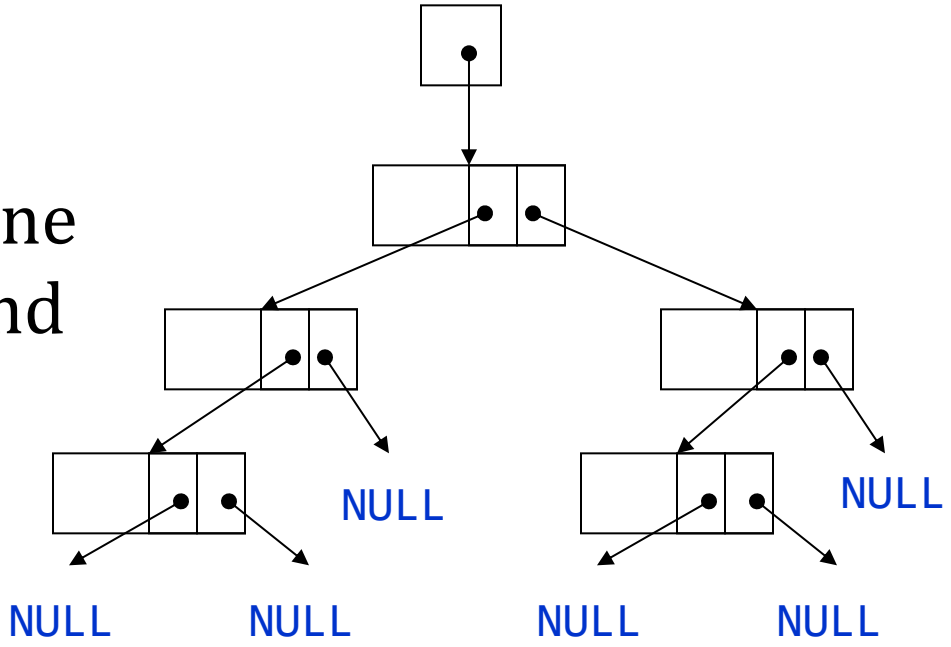
20.1

Definition and Application of Binary Trees

Definition and Application of Binary Trees

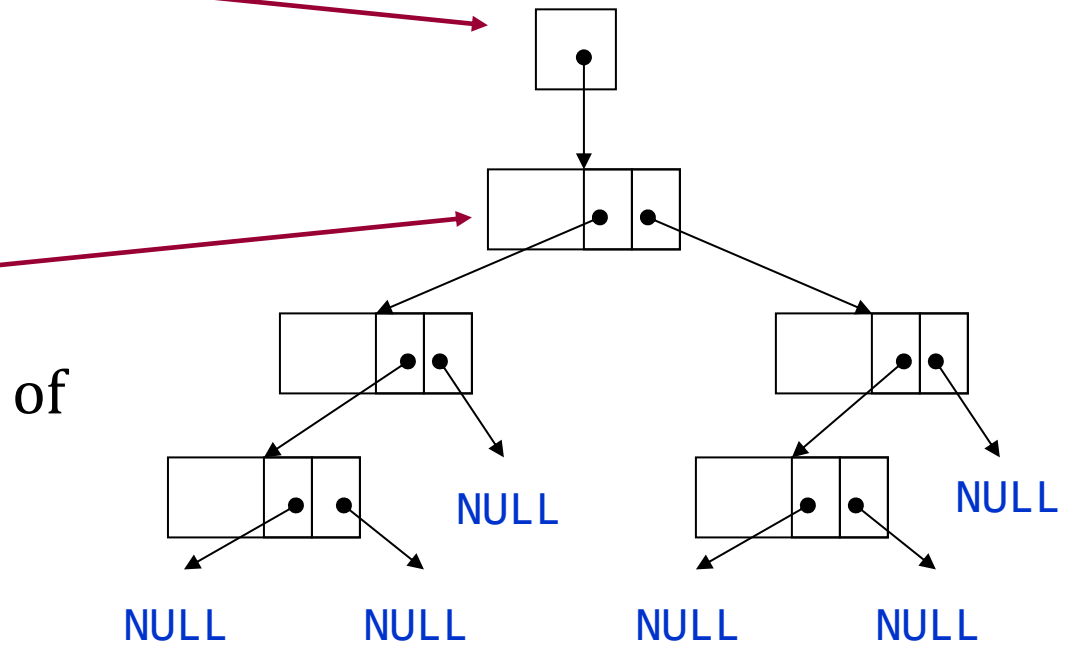
- Tree pointer
- Root node
- Parent
- Child (children)
- Leaf node
- Subtree (left/right)

- Binary tree:
  - a nonlinear linked list in which each node may point to 0, 1, or two other nodes
- Each node contains one or more data fields and two pointers



Binary Tree Terminology

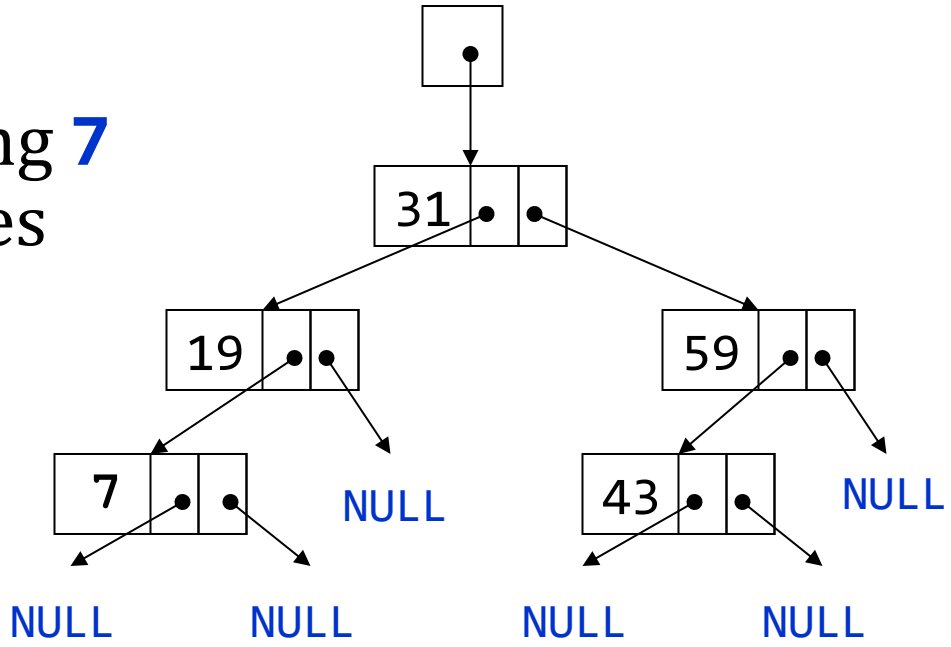
- Tree pointer:
  - like a head pointer for a linked list, it points to the first node in the binary tree
- Root node:
  - the node at the top of the tree



Binary Tree Terminology

- Leaf nodes:
  - nodes that have no children

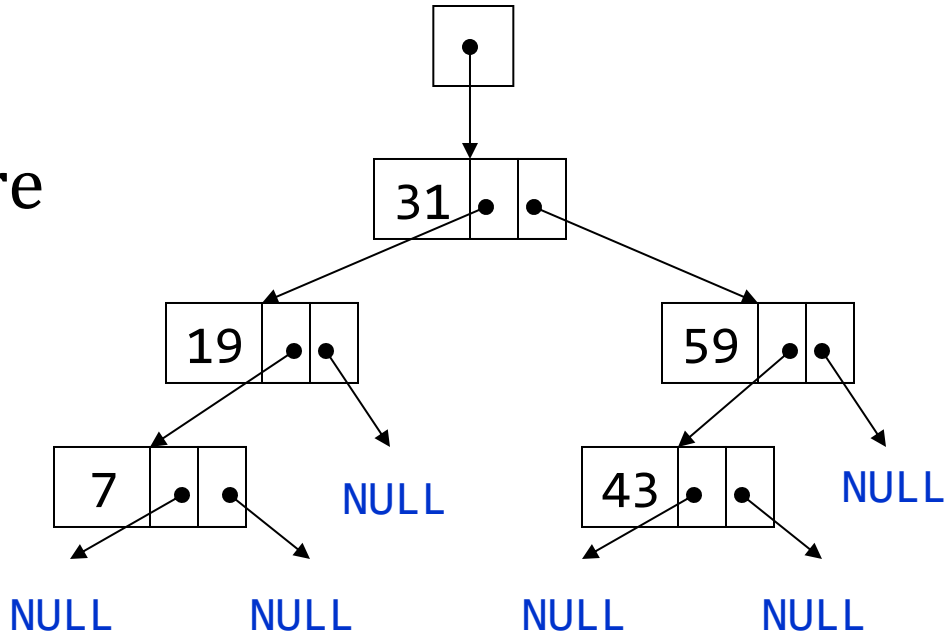
The nodes containing **7** and **43** are leaf nodes



Binary Tree Terminology

- Child nodes (children):
  - nodes below a given node

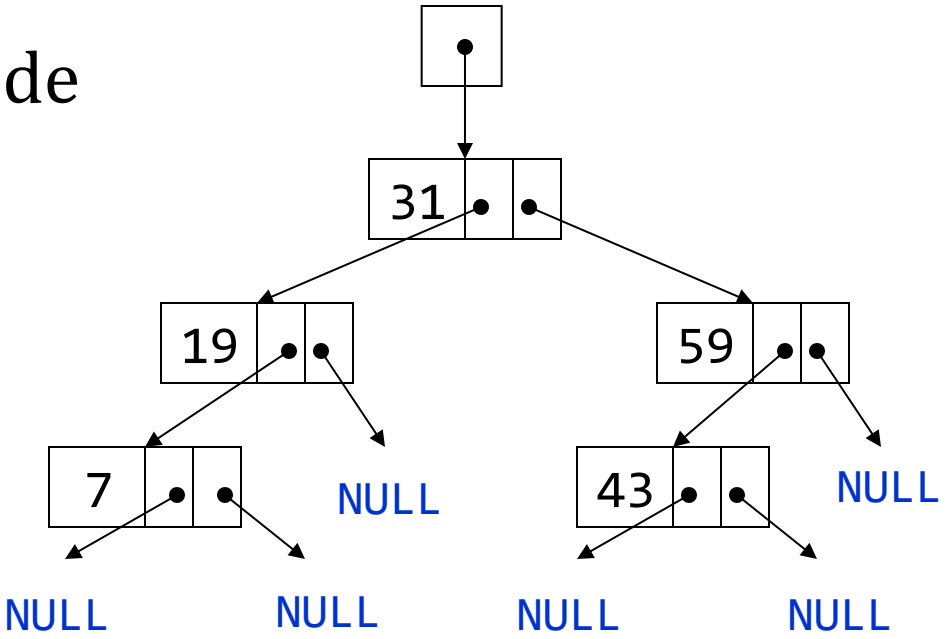
The children of the node containing **31** are the nodes containing **19** and **59**



Binary Tree Terminology

- Parent node:
  - node above a given node

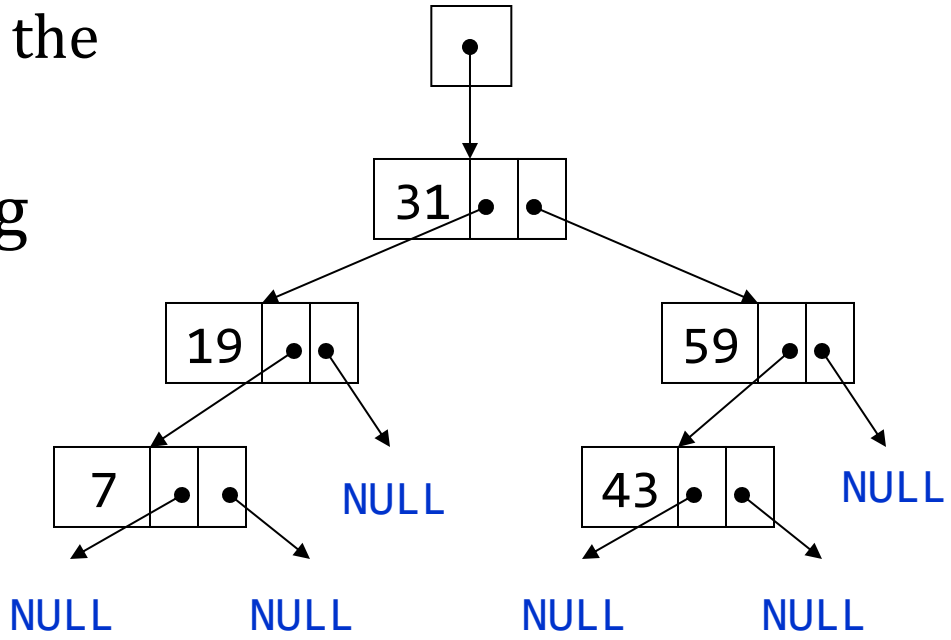
The parent of the node containing **43** is the node containing **59**



Binary Tree Terminology

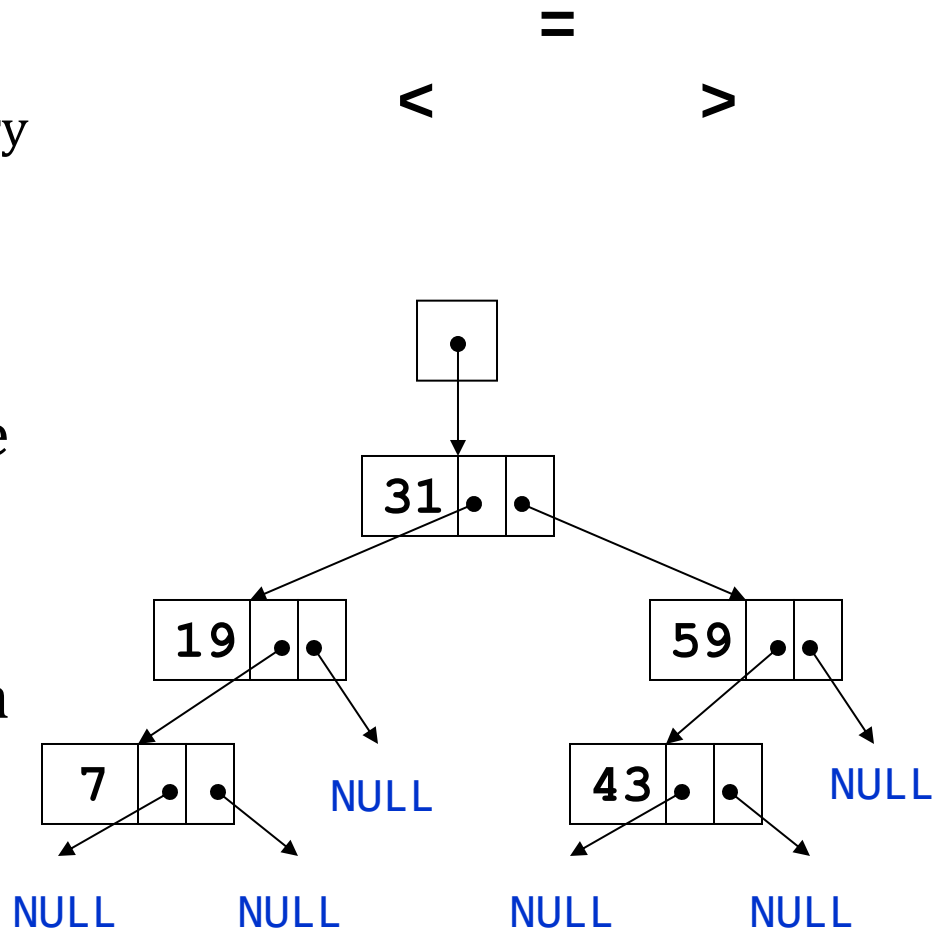
- Subtree:
  - the portion of a tree from a node down to the leaves

The nodes containing **19** and **7** are the left subtree of the node containing **31**



Uses of Binary Trees

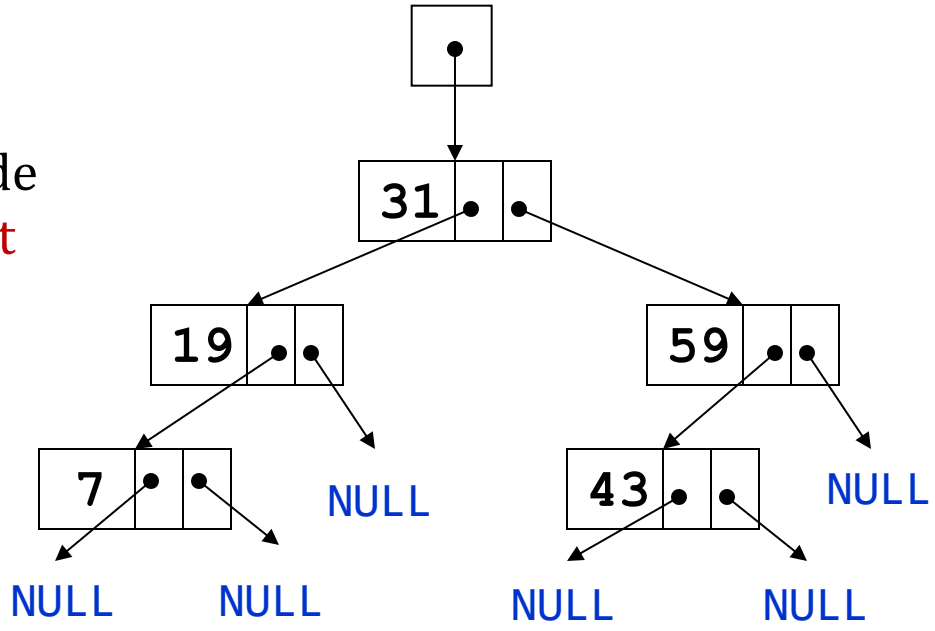
- Binary search tree:
  - data is organized in a binary tree to simplify (speed up) searches
- Left subtree of a node contains data values < the data in the node
- Right subtree of a node contains values > the data in the node



Searching in a Binary Tree

1) Start at root node
2) Examine node data:
   a) Is it the desired value (=)? Done
   b) Else, is the desired data < node data? Repeat step 2 with left subtree
   c) Else, is the desired data > node data? Repeat step 2 with right subtree
3) Continue until the desired value is found or a NULL pointer is reached

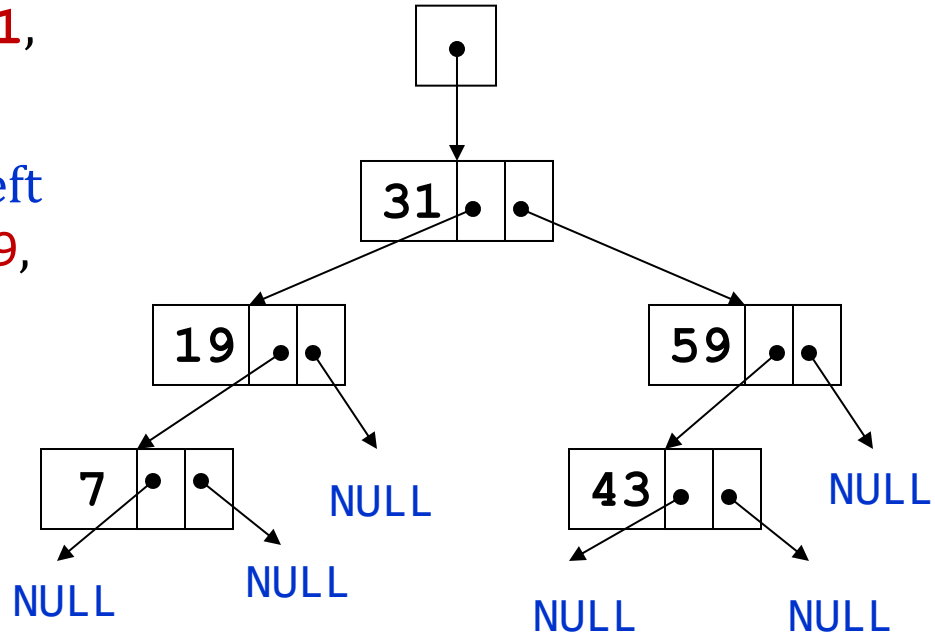
data stored in
ascending sequence

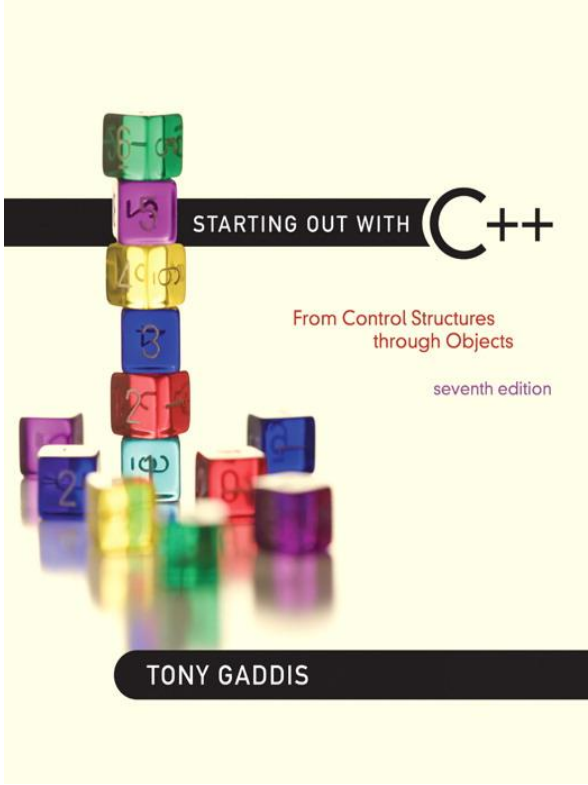


Searching in a Binary Tree

To locate the node containing **43**,

- Examine the root node (**31**) first
- Since 43 > 31, examine the right child of the node containing **31**, (**59**)
- Since 43 < 59, examine the left child of the node containing 59, (43)
- The node containing 43 has been found





20.2

Binary Search Tree **Operations**

Binary Search Tree Operations

- **Create** a binary search tree
  - organize data into a binary search tree
- **Insert** a node into a binary tree
  - put node into tree in its correct position to maintain order
- **Search** for a node in a binary tree
  - locate a node with particular data value
- **Delete** a node from a binary tree
  - remove a node and adjust links to maintain binary tree

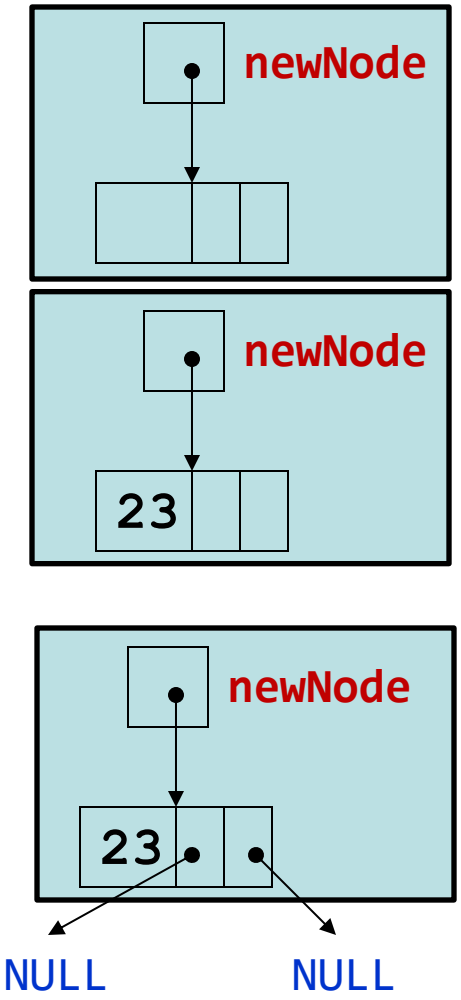
Binary Search Tree Node

- A node in a binary tree is like a node in a linked list with two node pointer fields:

```
struct TreeNode
{
   int value;
   TreeNode *left;
   TreeNode *right;
}
```

Creating a New Node

1. Allocate memory for new node:
   ```
   TreeNode *newNode = new TreeNode;
   ```
2. Initialize the contents of the node:
   ```
   newNode->value = num;
   ```
3. Set the pointers to NULL:
   ```
   newNode->Left
   = newNode->Right
   = NULL;
   ```



Inserting a Node in aBinary Search Tree

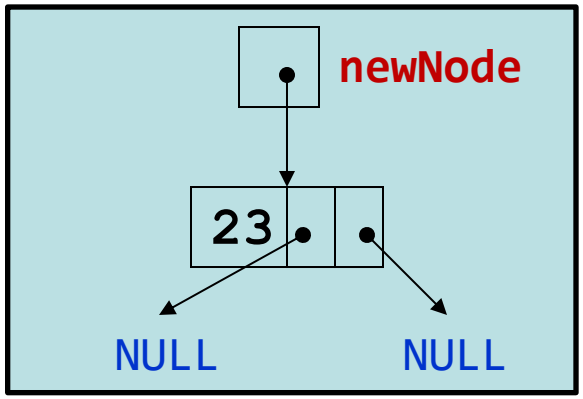
1) If the tree is empty, insert the new node as the root node

2) Else, compare new node against left or right child, depending on whether data value of new node is < or > root node

3) Continue comparing and choosing left or right subtree until the NULL pointer is found

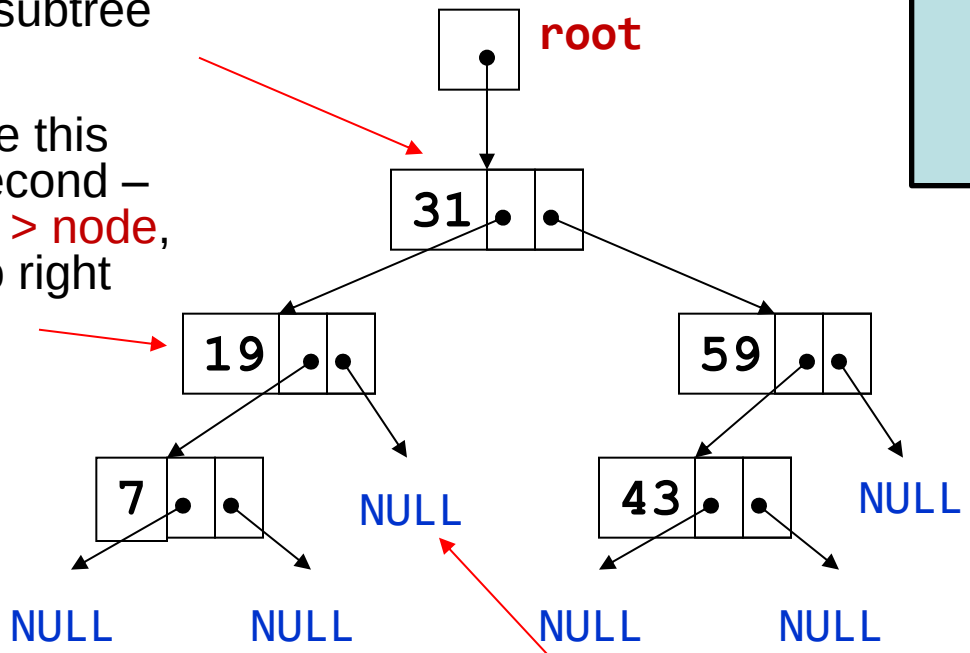
4) Set this NULL pointer to point to the new node

Inserting a Node in a Binary Search Tree



Examine this node first – value is < node, so go to left subtree

Examine this node second – value is > node, so go to right subtree



Since the right subtree is NULL, insert here

Traversing a Binary Tree

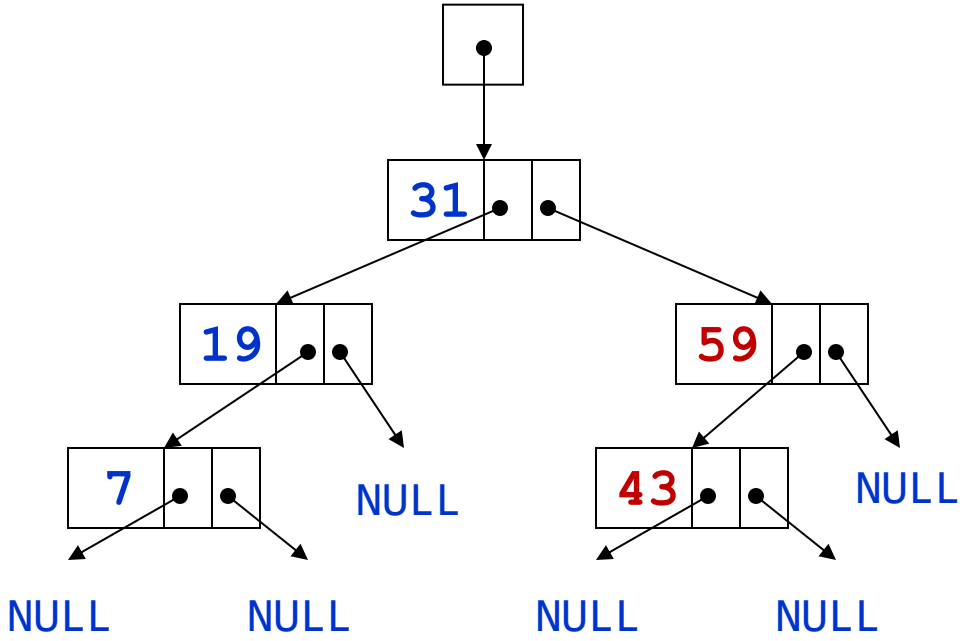
Three traversal methods:

1) Inorder: left / data / right
   a) Traverse left subtree of node
   b) Process data in node
   c) Traverse right subtree of node

2) Preorder: data / left / right
   a) Process data in node
   b) Traverse left subtree of node
   c) Traverse right subtree of node

3) Postorder: left / right / data
   a) Traverse left subtree of node
   b) Traverse right subtree of node
   c) Process data in node

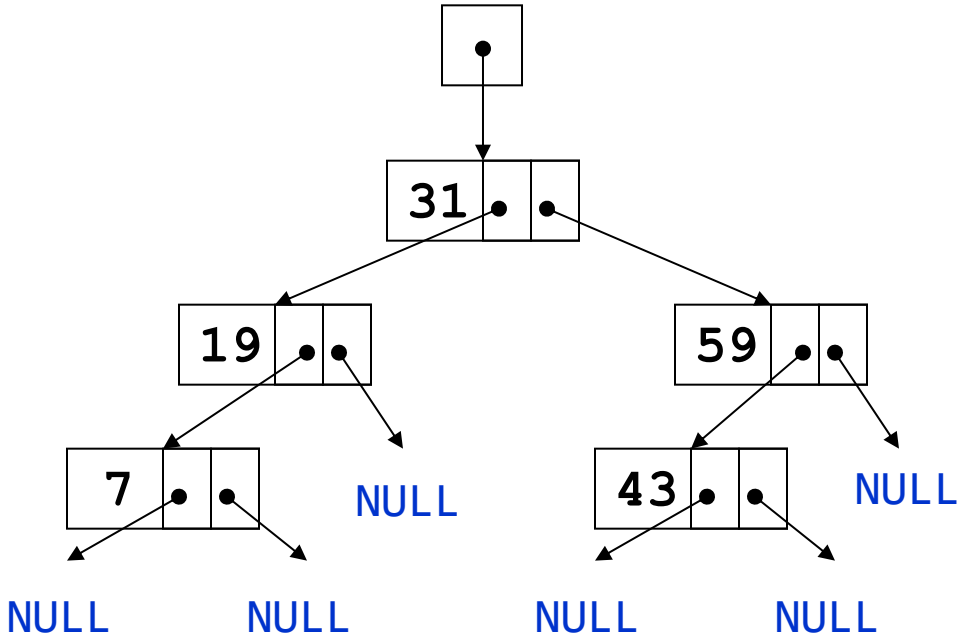
Traversing a Binary Tree



TRAVERSAL METHOD	NODES VISITED IN ORDER
Inorder L / D / R	7, 19, 31, 43, 59
Preorder D / L / R	31, 19, 7, 59, 43
Postorder L / R / D	7, 19, 43, 59, 31

Searching in a Binary Tree

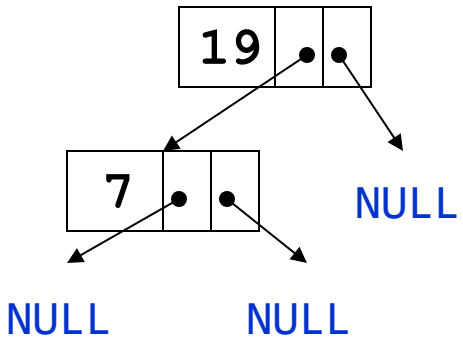
- Start at the root node, traverse the tree looking for a value
- Stop when the value is found or a `NULL` pointer is detected
- Can be implemented as a `bool` function



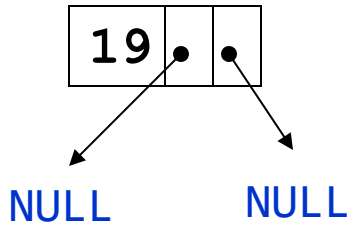
Search for `43` -> returns `true`
Search for `17` -> returns `false`

Deleting a Node from a Binary Tree – Leaf Node

- If the node to be deleted is a leaf node, replace the parent node's pointer to it with a NULL pointer, then delete the node



Deleting node with 7
– before deletion

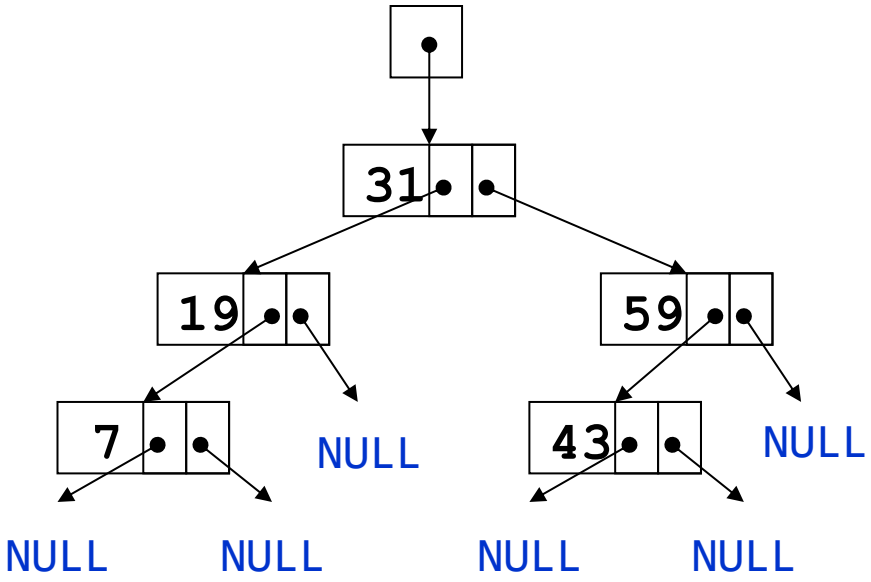


Deleting node with 7
– after deletion

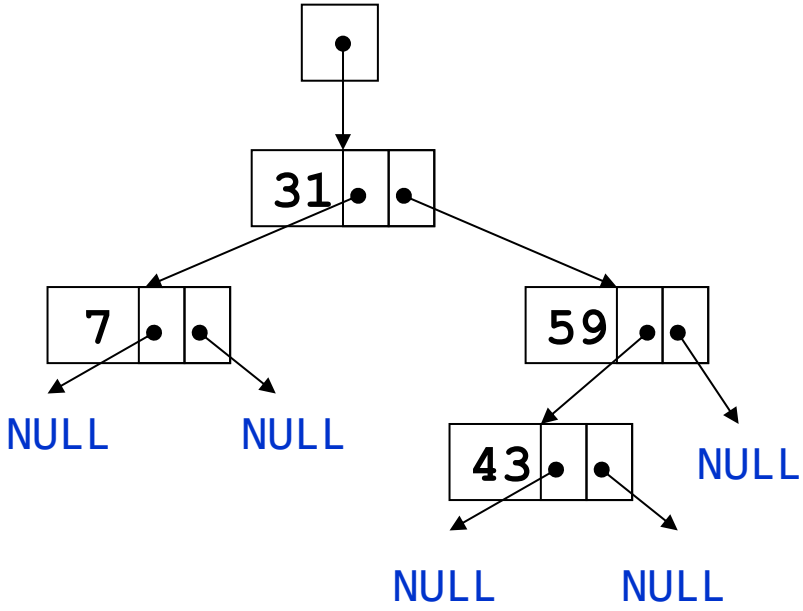
Deleting a Node from a Binary Tree – One Child

- If the node to be deleted has one child node, adjust the pointers so that the parent of the node to be deleted points to the child of node to be deleted, then delete the node

Deleting a Node from a Binary Tree – One Child



Deleting node with 19
– before deletion

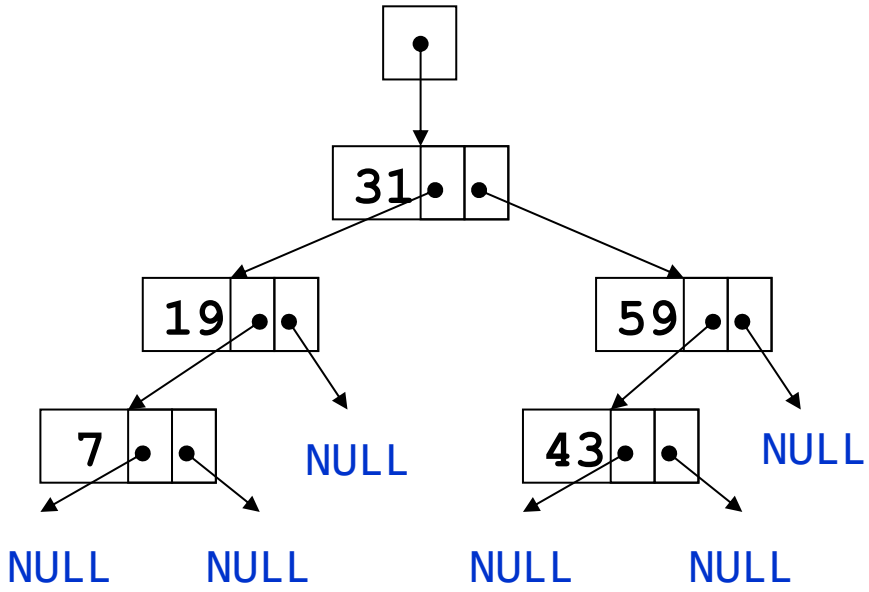


Deleting node with 19
– after deletion

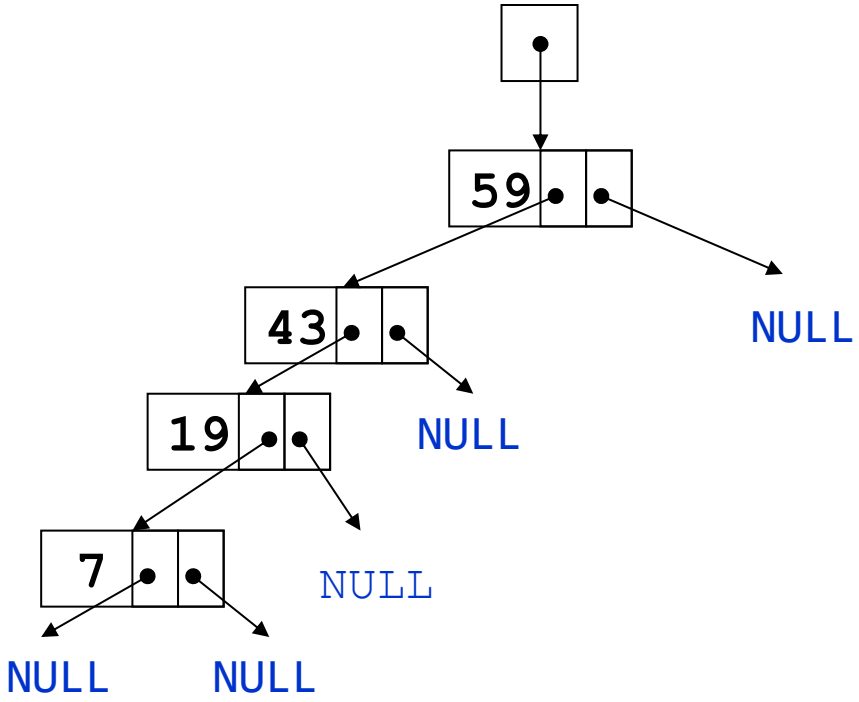
Deleting a Node from a Binary Tree – Two Children

- If the node to be deleted has both left and right children,
  - 'Promote' one child to take the place of the deleted node
  - Locate correct position for other child in subtree of promoted child
- The textbook convention: promote the right child, then position the left subtree underneath

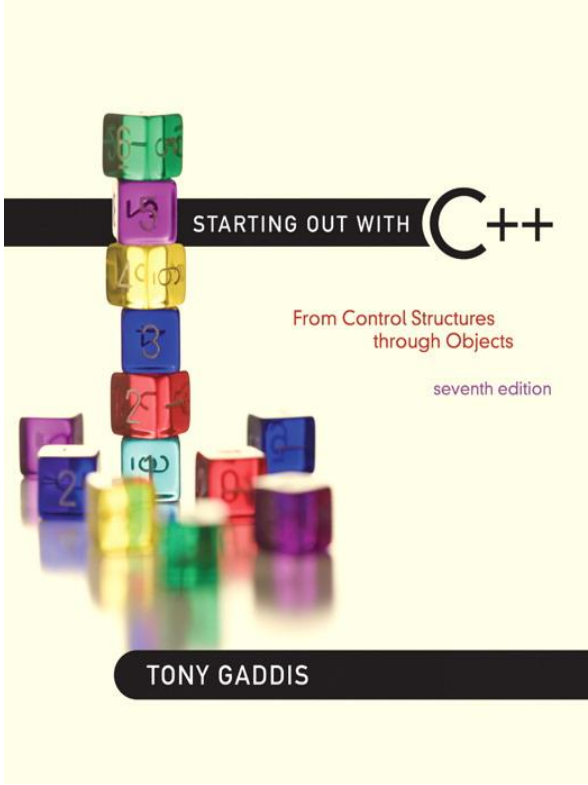
Deleting a Node from a Binary Tree – Two Children



Deleting node with 31
– before deletion



Deleting node with 31
– after deletion



20.3

Template Considerations for Binary Search Trees

Template Considerations for Binary Search Trees

- A binary tree can be implemented as a template allowing flexibility in determining the type of data stored

- The implementation must support relational operators **>**, **<**, and **==** to allow comparison of nodes