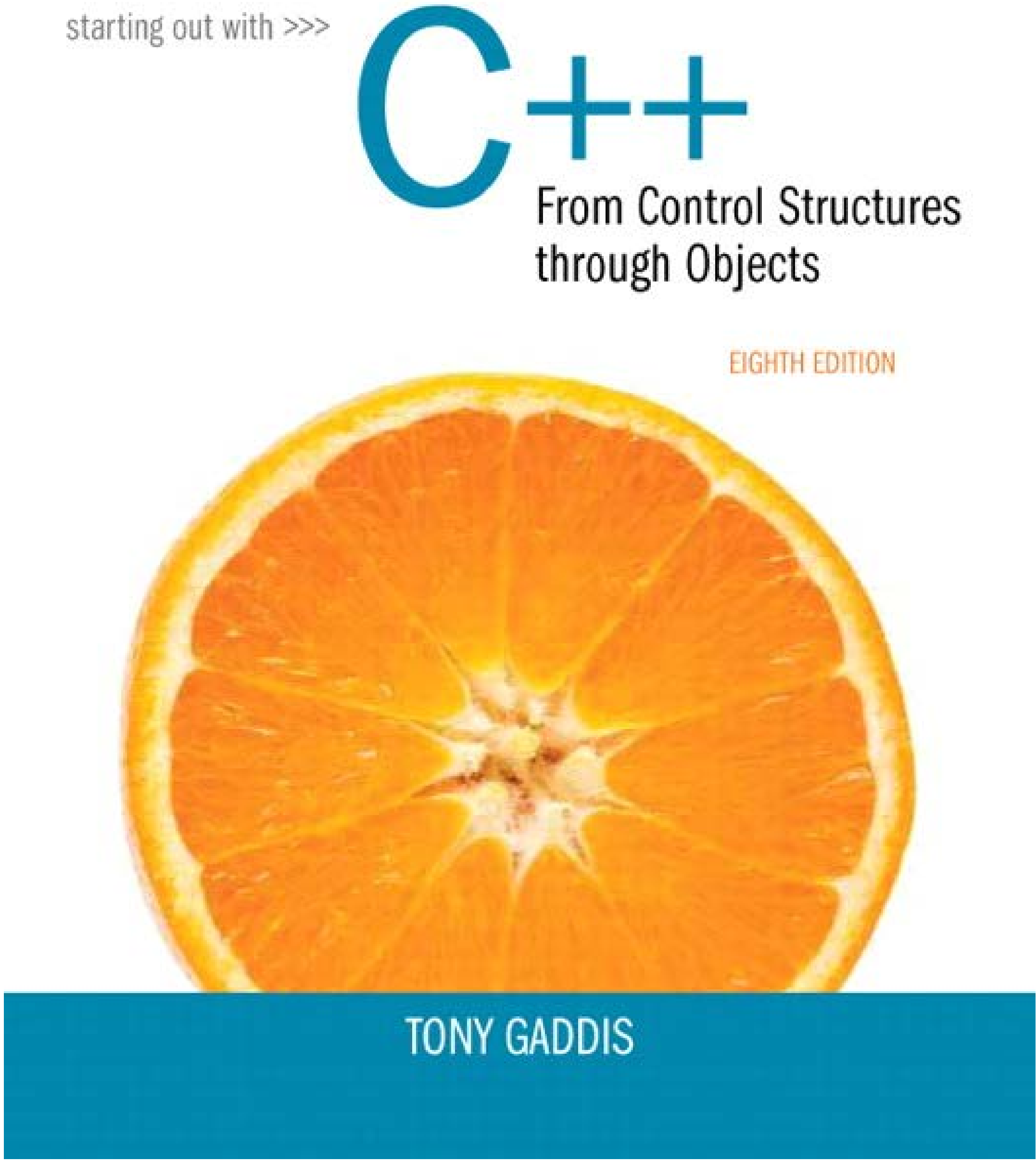
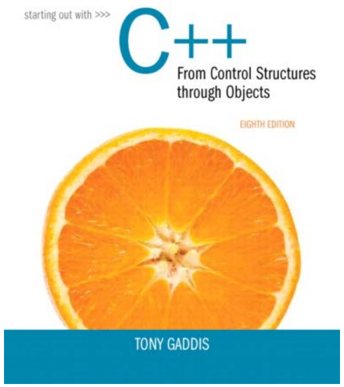
Chapter 20:

Binary Trees



Addison-Wesley
is an imprint of
PEARSON



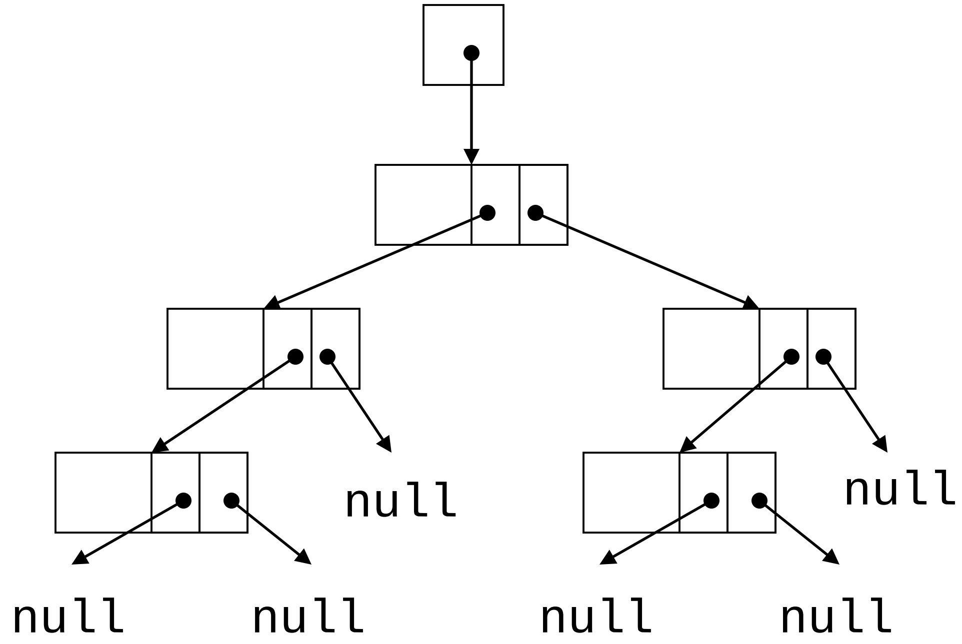
20.1

Definition and Application of Binary Trees

Addison-Wesley is an imprint of PEARSON

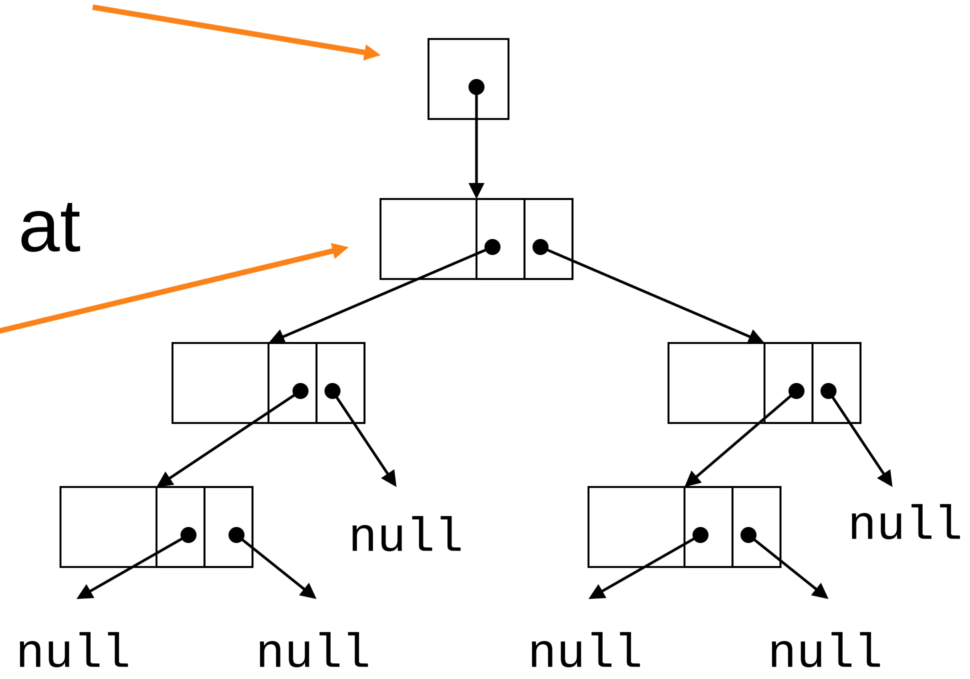
Definition and Application of Binary Trees

- 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

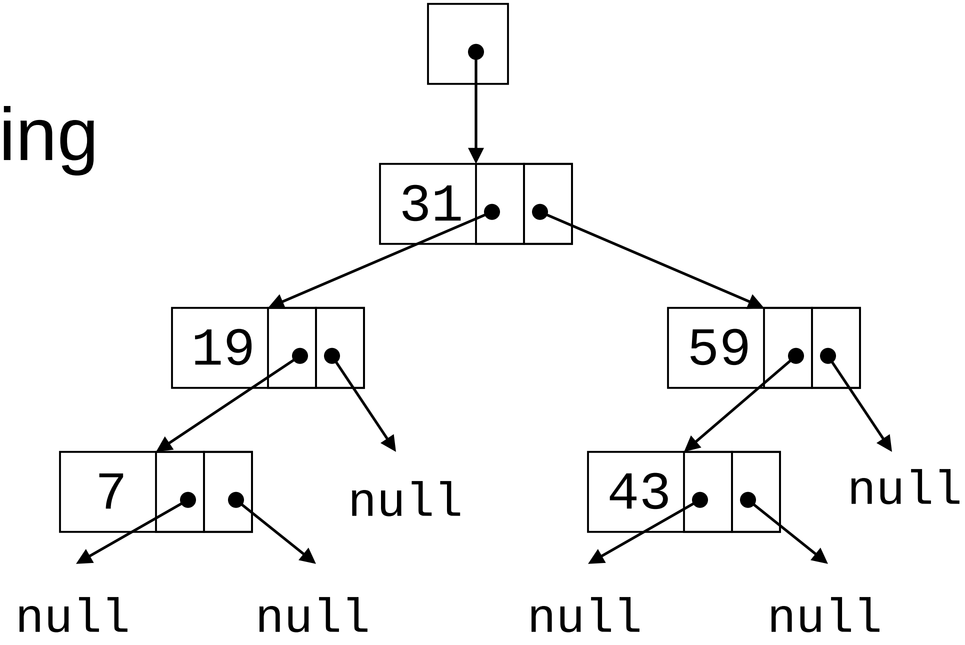
- Tree pointer: like a head pointer for a linked list, it points to the first node in the binary tree
- Root node: the node at the top of the tree



Binary Tree Terminology

- Leaf nodes: nodes that have no children

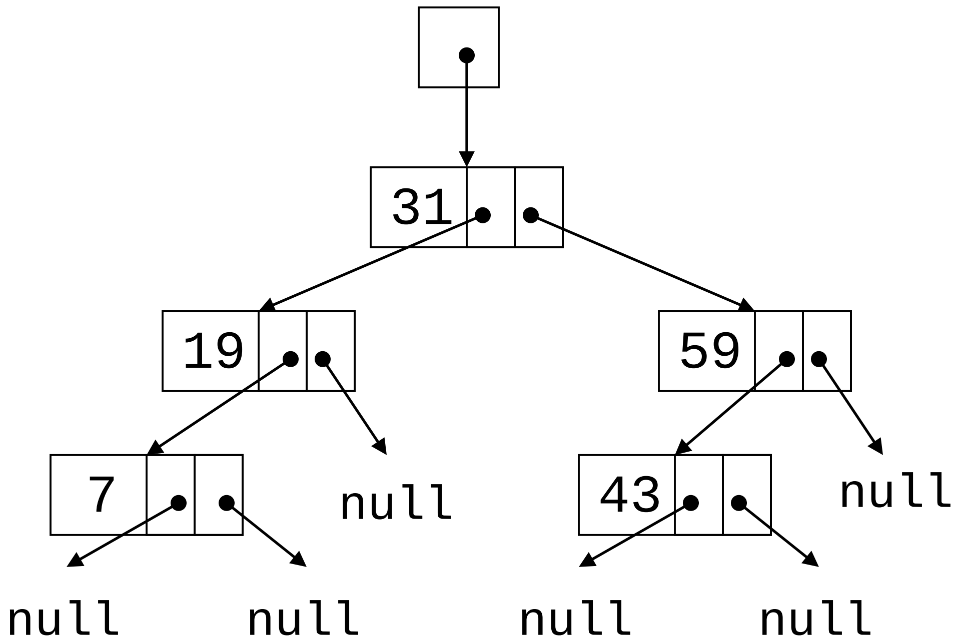
  The nodes containing 7 and 43 are leaf nodes



Binary Tree Terminology

- Child nodes, children: nodes below a given node

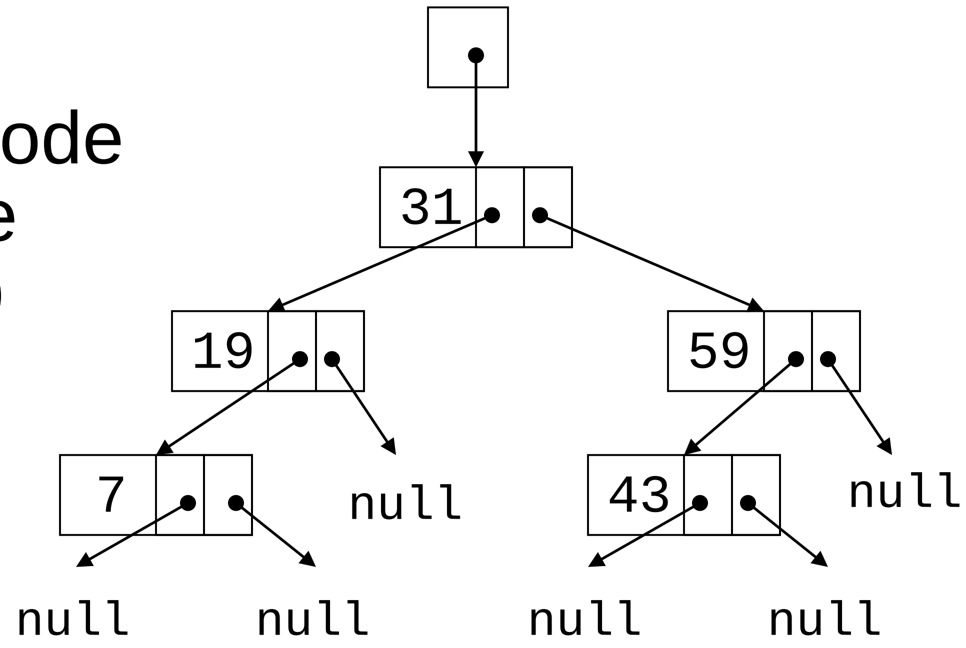
  The children of the node containing 31 are the nodes containing 19 and 59



Binary Tree Terminology

- Parent node: node above a given node

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

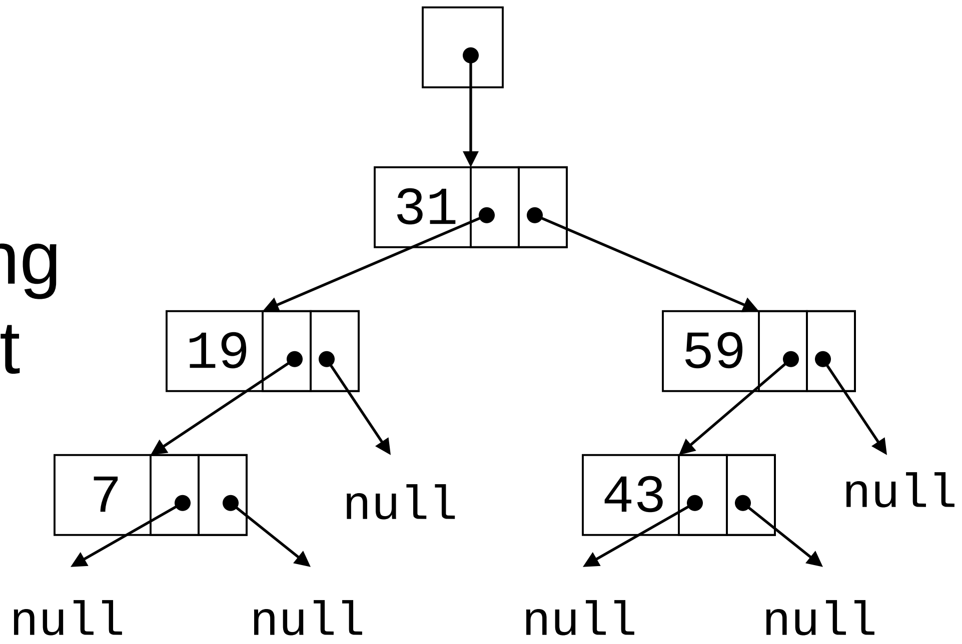


Addison-Wesley is an imprint of PEARSON

Binary Tree Terminology

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

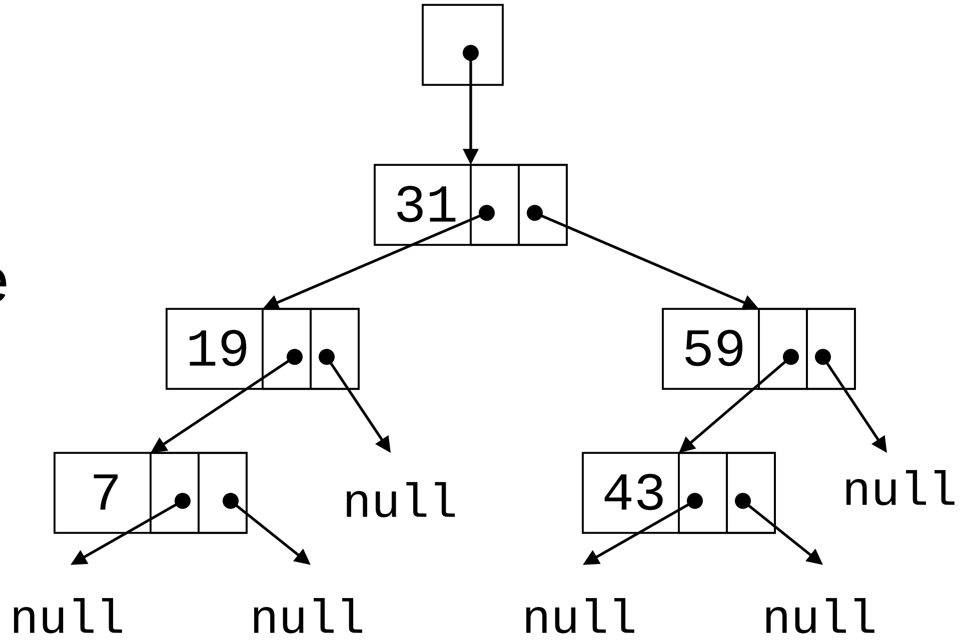
  The nodes containing 19 and 7 are the left subtree of the node containing 31



Addison-Wesley is an imprint of PEARSON

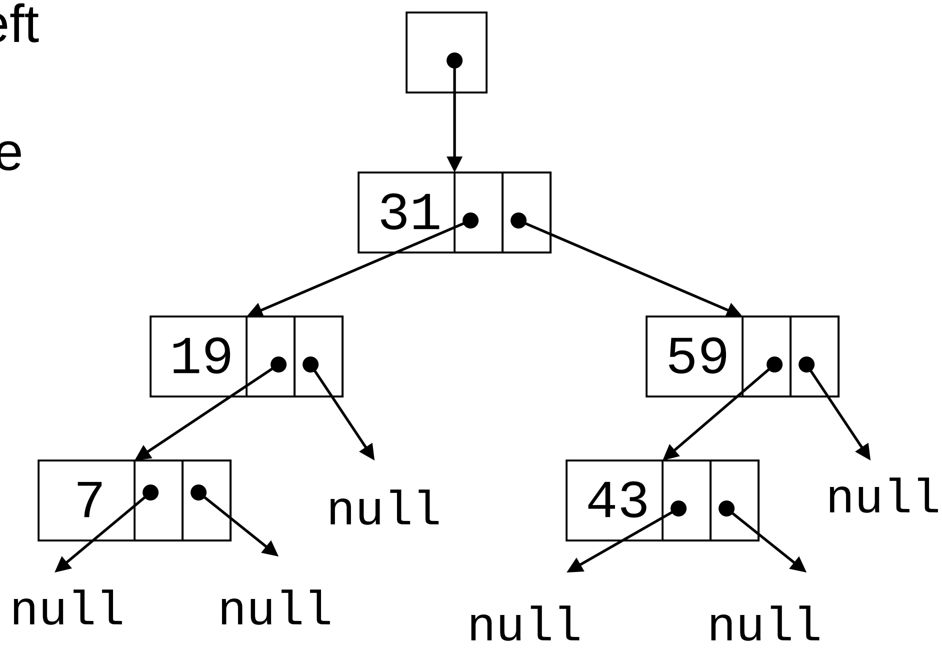
Uses of Binary Trees

- Binary search tree: data organized in a binary tree to simplify 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

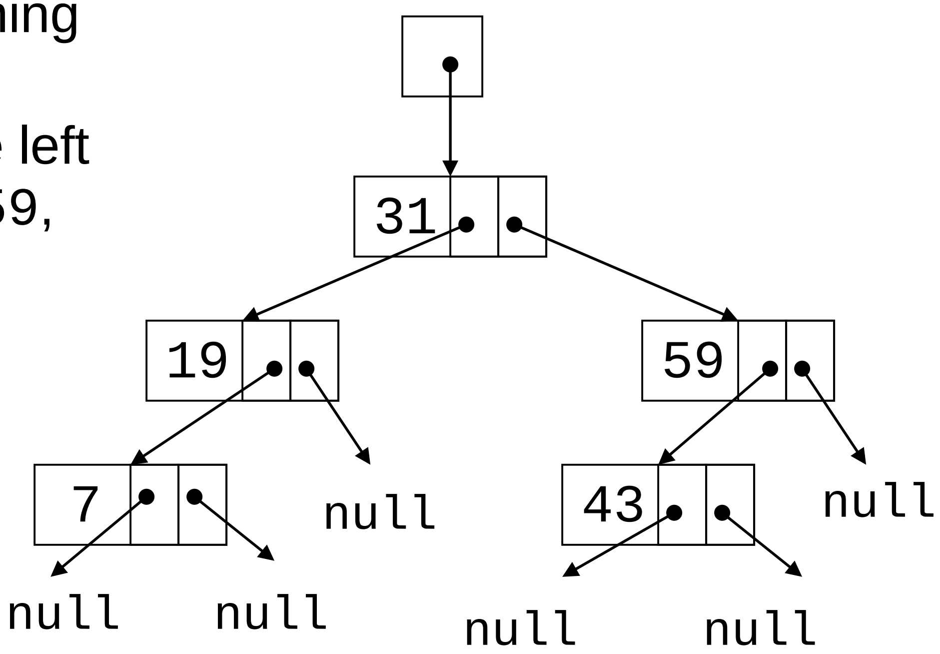
1) Start at root node
2) Examine node data:
    a) Is it desired value? Done
    b) Else, is desired data < node data? Repeat step 2 with left subtree
    c) Else, is desired data > node data? Repeat step 2 with right subtree
3) Continue until desired value found or a null pointer reached



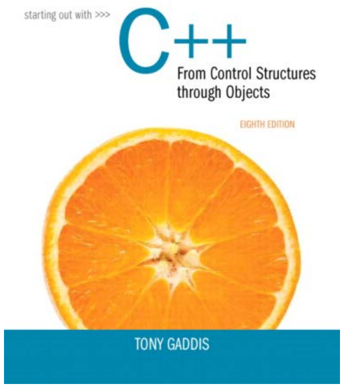
Searching in a Binary Tree

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



Addison-Wesley
is an imprint of
PEARSON



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
- Find 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

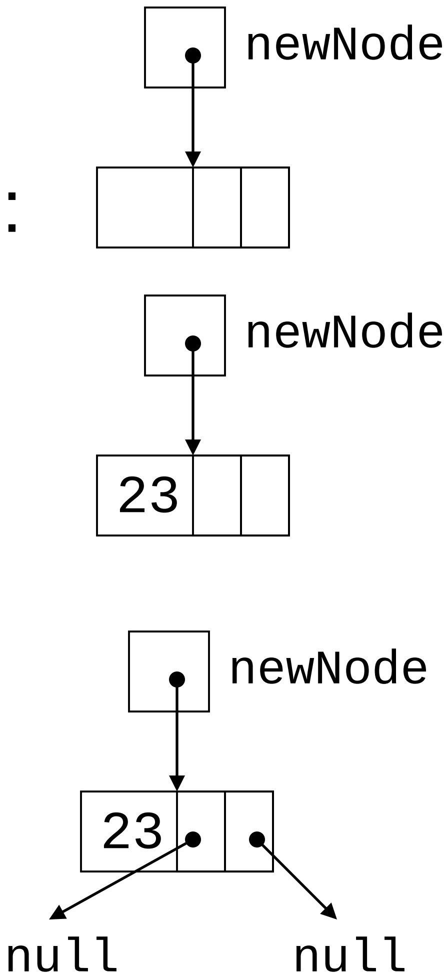
Binary Search Tree Node

- 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

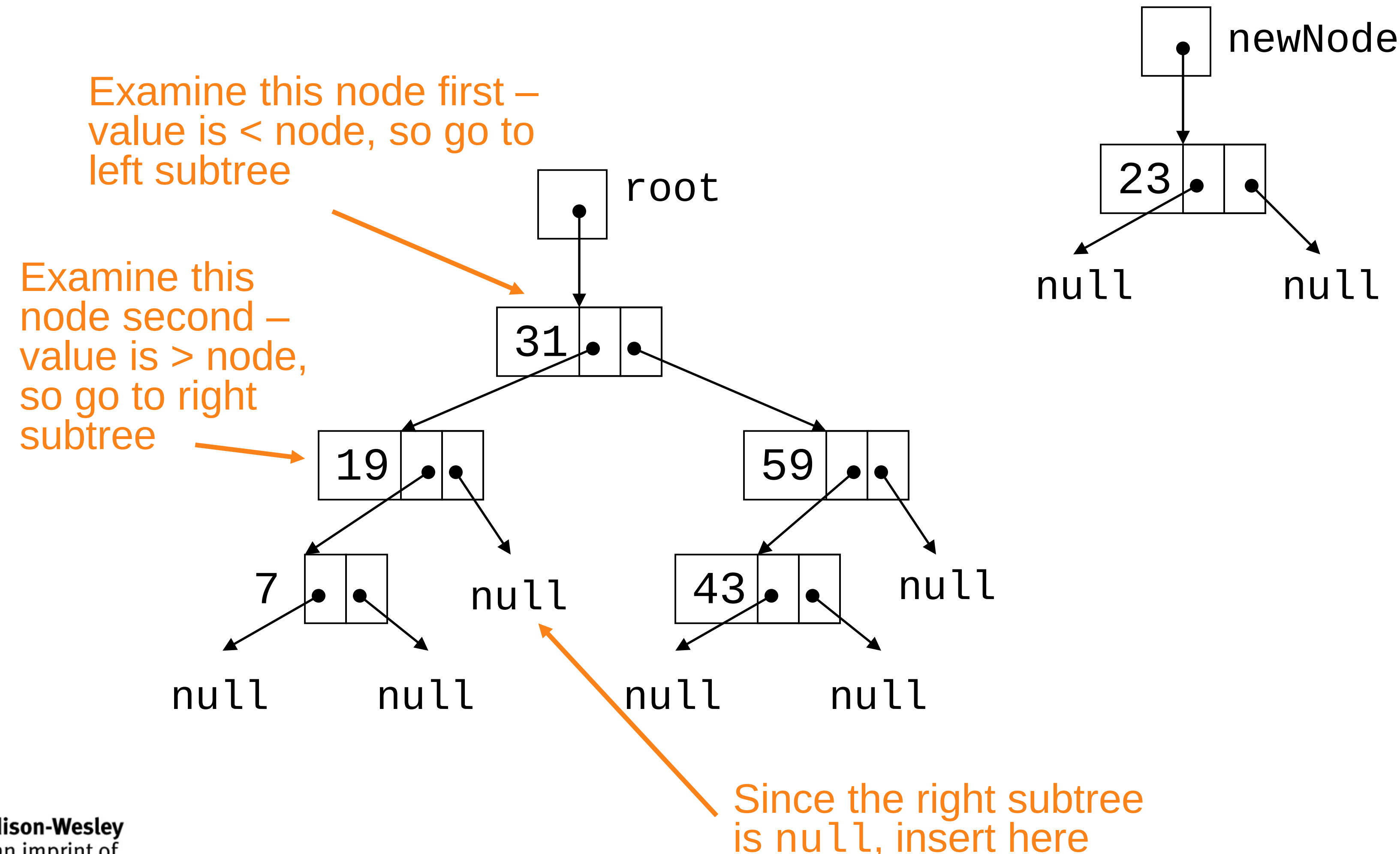
- Allocate memory for new node:
  `newNode = new TreeNode;`
- Initialize the contents of the node:
  `newNode->value = num;`
- Set the pointers to `nullptr`:
  ```
  newNode->Left
  = newNode->Right
  = nullptr;
  ```



Inserting a Node in a Binary Search Tree

1) If 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 unitl null pointer found
4) Set this null pointer to point to new node

Inserting a Node in a Binary Search Tree

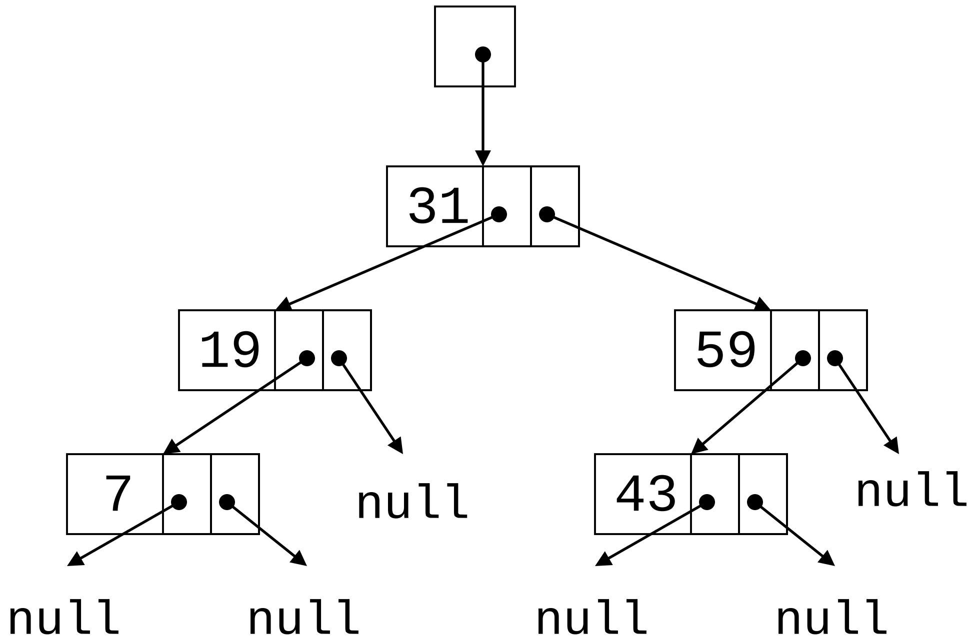


Traversing a Binary Tree

Three traversal methods:

1) Inorder:
   a) Traverse left subtree of node
   b) Process data in node
   c) Traverse right subtree of node
2) Preorder:
   a) Process data in node
   b) Traverse left subtree of node
   c) Traverse right subtree of node
3) Postorder:
   a) Traverse left subtree of node
   b) Traverse right subtree of node
   c) Process data in node

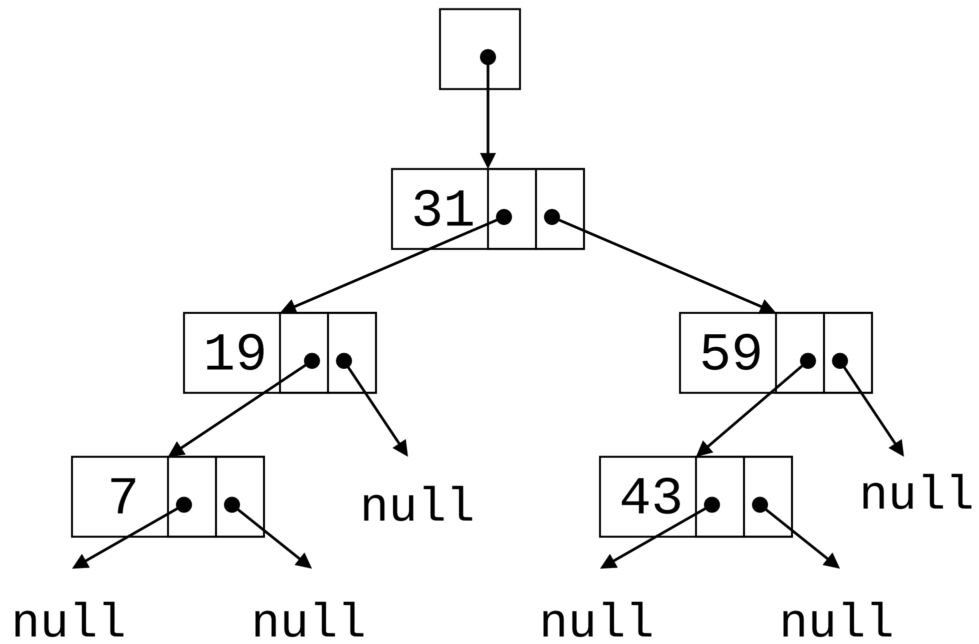
Traversing a Binary Tree



TRAVERSAL METHOD	NODES VISITED IN ORDER
Inorder	7, 19, 31, 43, 59
Preorder	31, 19, 7, 59, 43
Postorder	7, 19, 43, 59, 31

Searching in a Binary Tree

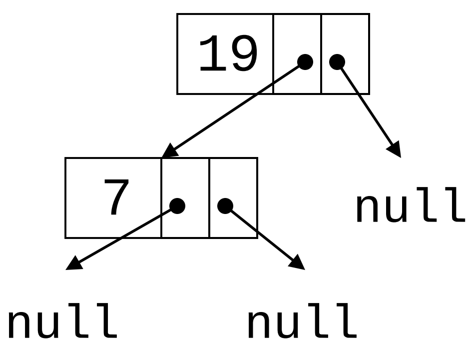
- Start at root node, traverse the tree looking for value
- Stop when value found or null pointer detected
- Can be implemented as a `bool` function



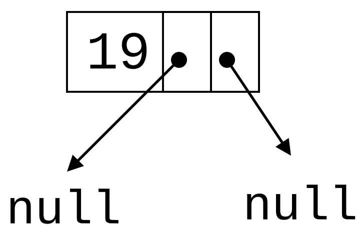
Search for 43? return `true`
Search for 17? return `false`

Deleting a Node from a Binary Tree – Leaf Node

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



Deleting node with 7 – before deletion

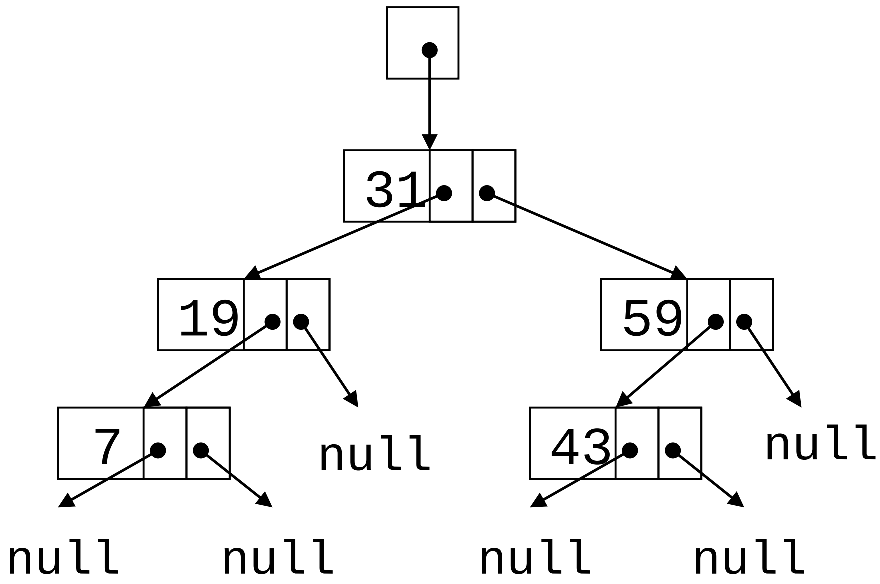


Deleting node with 7 – after deletion

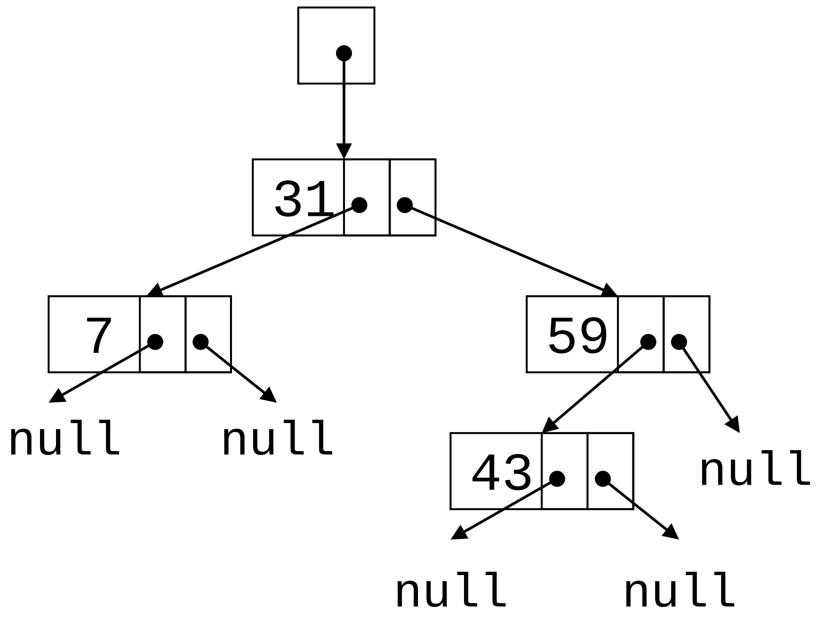
Deleting a Node from a Binary Tree – One Child

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



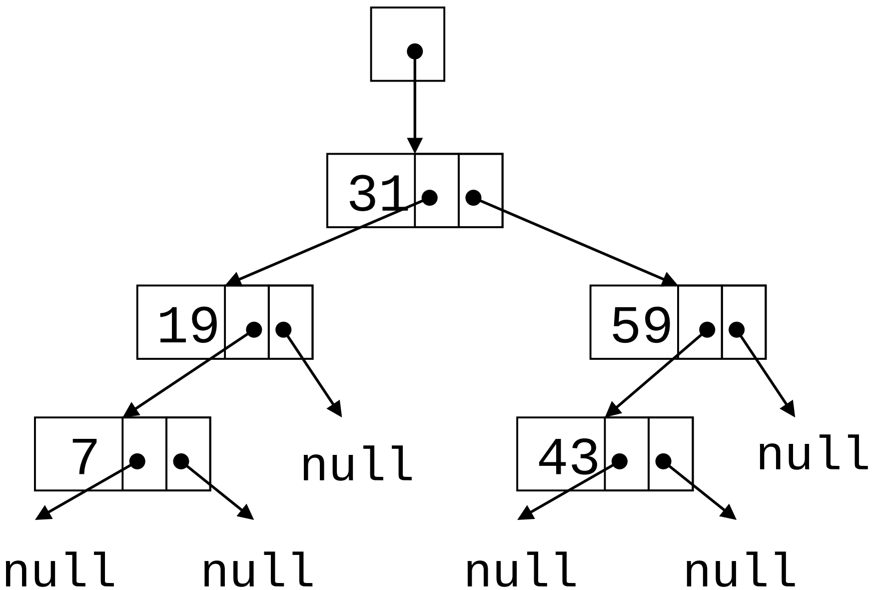
Deleting node with 19
– after deletion

Deleting a Node from a Binary Tree – Two Children

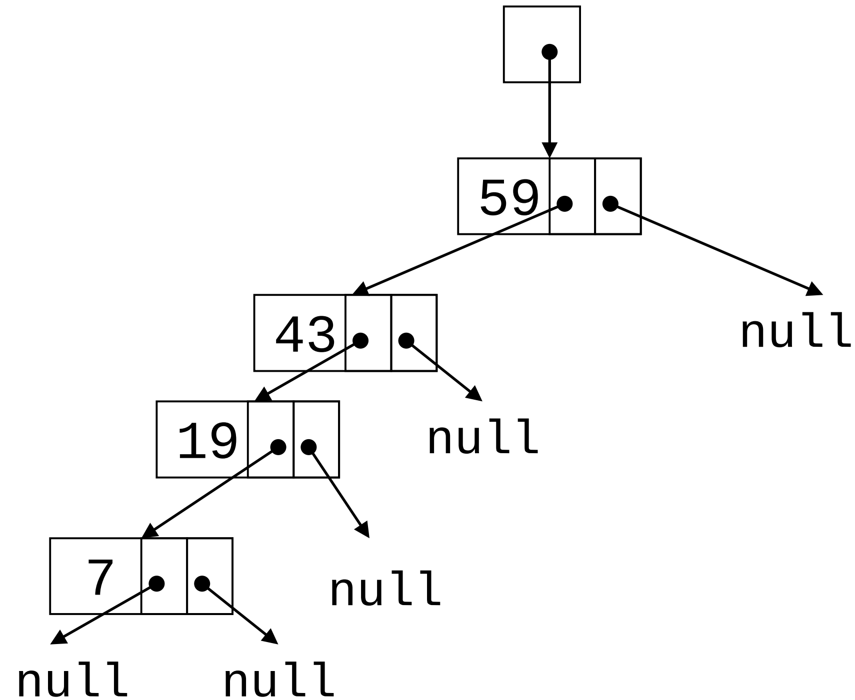
- If node to be deleted has left and right children,
  - ‘Promote’ one child to take the place of the deleted node
  - Locate correct position for other child in subtree of promoted child
- Convention in text: promote the right child, position left subtree underneath

Addison-Wesley is an imprint of PEARSON

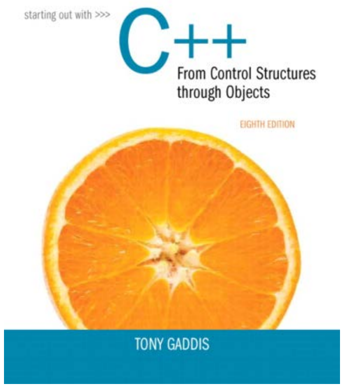
Deleting a Node from a Binary Tree – Two Children



Deleting node with 31
– before deletion



Deleting node with 31
– after deletion



20.3

Template Considerations for Binary Search Trees

Addison-Wesley is an imprint of PEARSON

Template Considerations for Binary Search Trees

- Binary tree can be implemented as a template, allowing flexibility in determining type of data stored
- Implementation must support relational operators >, <, and == to allow comparison of nodes

Addison-Wesley
is an imprint of
PEARSON