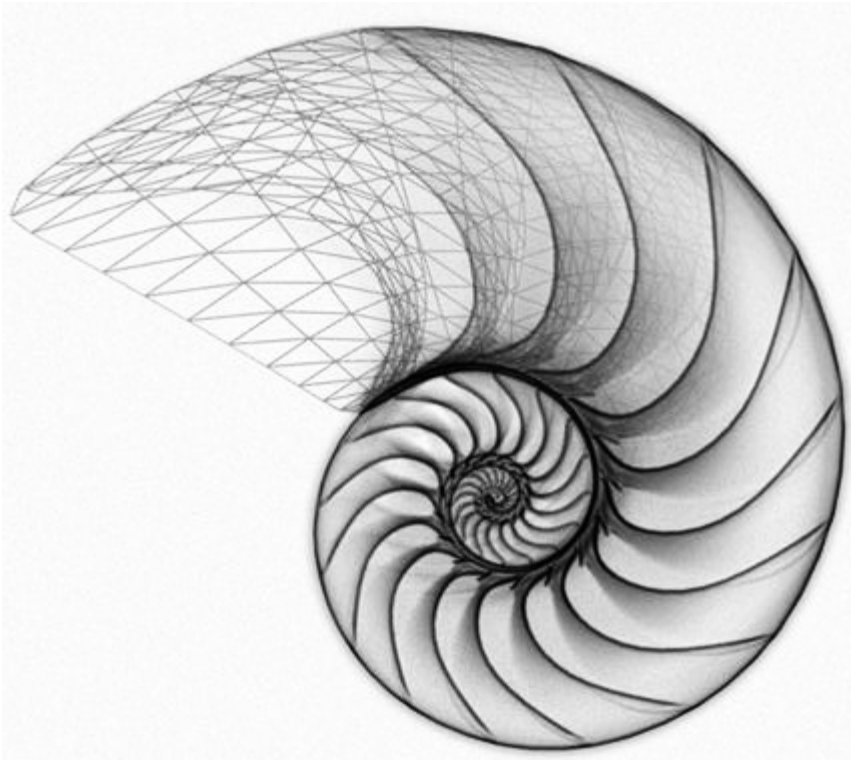


Lecture 15

Binary Search Tree

CSE225: Data Structures and Algorithms

Binary Search Tree Specification

Structure:	**The placement of each element in the binary tree must satisfy the binary search property: The value of the key of an element is greater than the value of the key of any element in its left subtree, and less than the value of the key of any element in its right subtree.**
Operations:	
MakeEmpty	
Function	Initializes tree to empty state.
Postcondition	Tree exists and is empty.
Boolean IsEmpty	
Function	Determines whether tree is empty.
Postcondition	Returns true if tree is empty and false otherwise.
Boolean IsFull	
Function	Determines whether tree is full.
Postcondition	Returns true if tree is full and false otherwise.

Binary Search Tree Specification

int LengthIs	
Function	Determines the number of elements in tree.
Postcondition	Returns the number of elements in tree.
RetrieveItem(ItemType& item. Boolean& found)	
Function	Retrieves item whose key matches item's key (if present).
Precondition	Key member of item is initialized.
Postcondition	If there is an element someItem whose key matches item's key, then found = true and item is a copy of someItem; otherwise, found = false and item is unchanged. Tree is unchanged.
InsertItem(ItemType item)	
Function	Adds item to tree.
Precondition	Tree is not full. item is not in tree.
Postcondition	item is in tree. Binary search property is maintained.

Binary Search Tree Specification

DeleteItem(ItemType item)	
Function	Deletes the element whose key matches item's key.
Precondition	Key member of item is initialized. One and only one element in tree has a key matching item's key.
Postcondition	No element in tree has a key matching item's key.
Print()	
Function	Prints the values in the tree in ascending key order.
Postcondition	Items in the tree are printed in ascending key order.

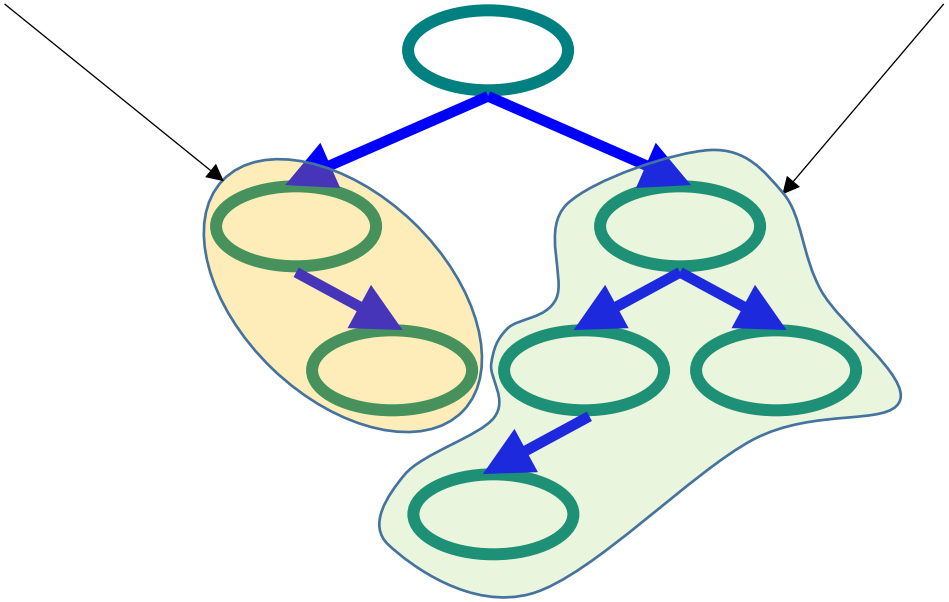
Binary Search Tree Specification

ResetTree(OrderType order)	
Function	Initializes current position for an iteration through the tree in OrderType order.
Postcondition	Current position is prior to root of tree.
GetNextItem(ItemType& item, OrderType order, Boolean& finished)	
Function	Gets the next element in tree.
Precondition	Current position is defined. Element at current position is not last in tree.
Postcondition	Current position is one position beyond current position at entry to GetNextItem. finished = (current position is last in tree). item is a copy of element at current position.

Function `LengthIs`

- Calculate the number of nodes in the tree recursively
- If tree is empty (base case)
  - length = 0
- If tree is non-empty (general case)
  - Length = number of nodes in the left subtree + number of nodes in the right subtree + 1

Number of nodes in left subtree + 1 + Number of nodes in right subtree

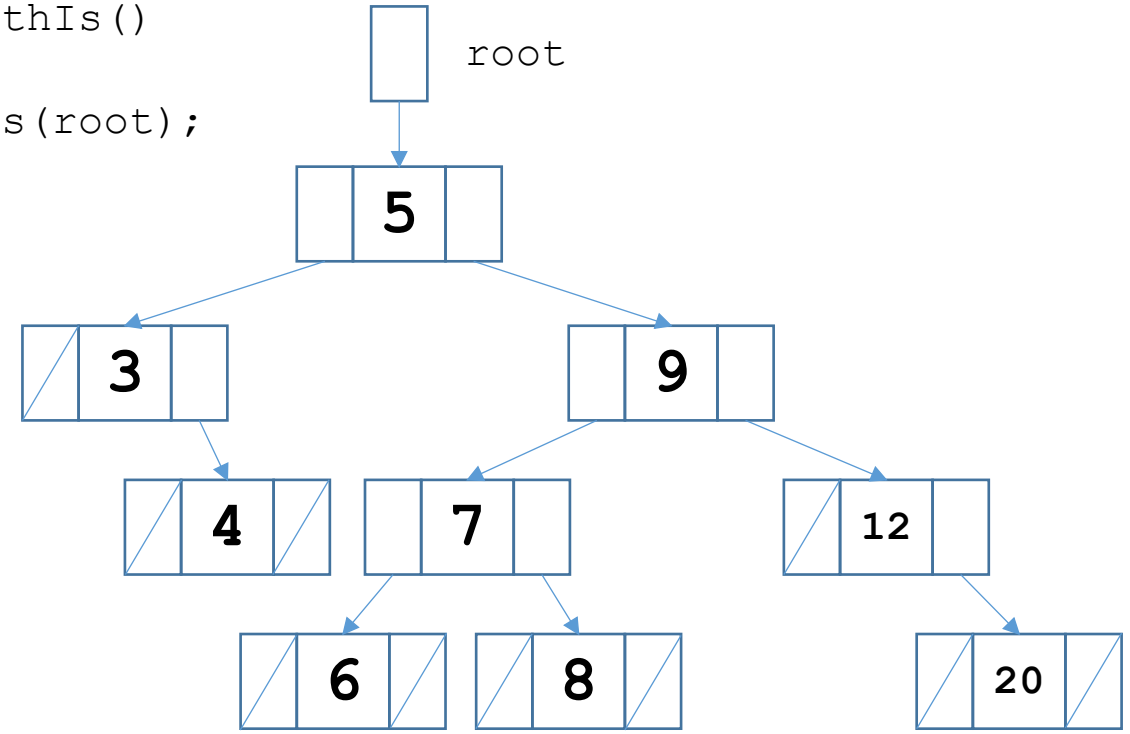


Function `LengthIs`

```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}

int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

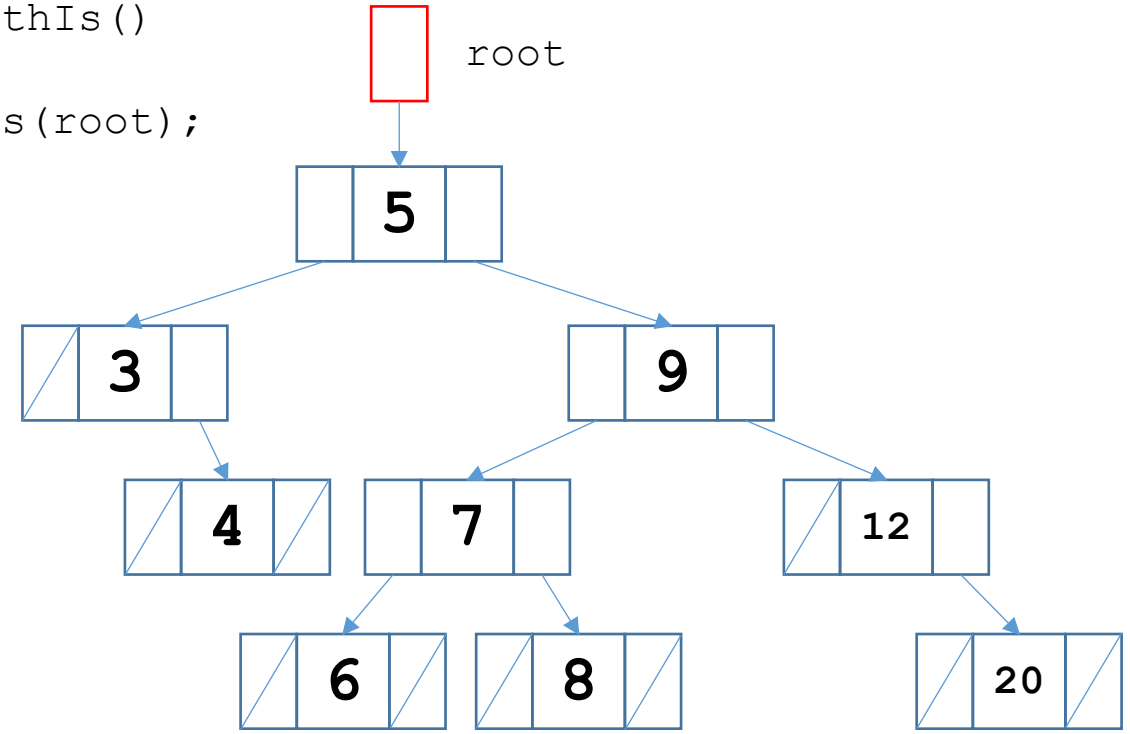
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



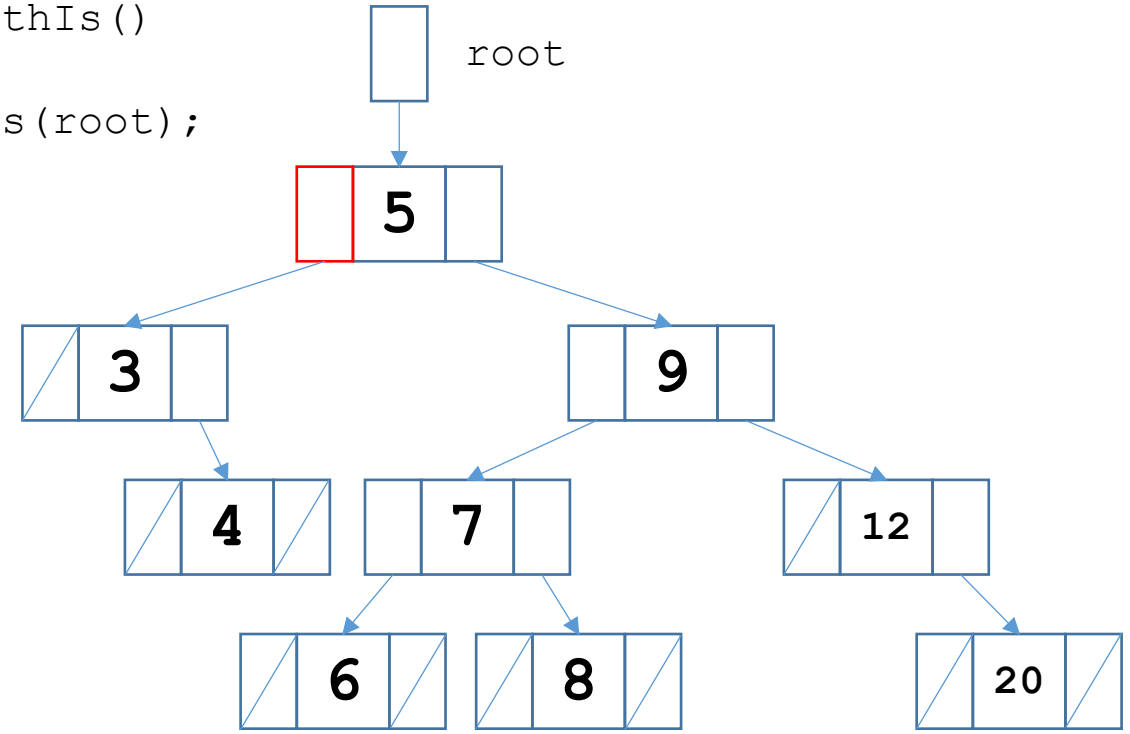
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



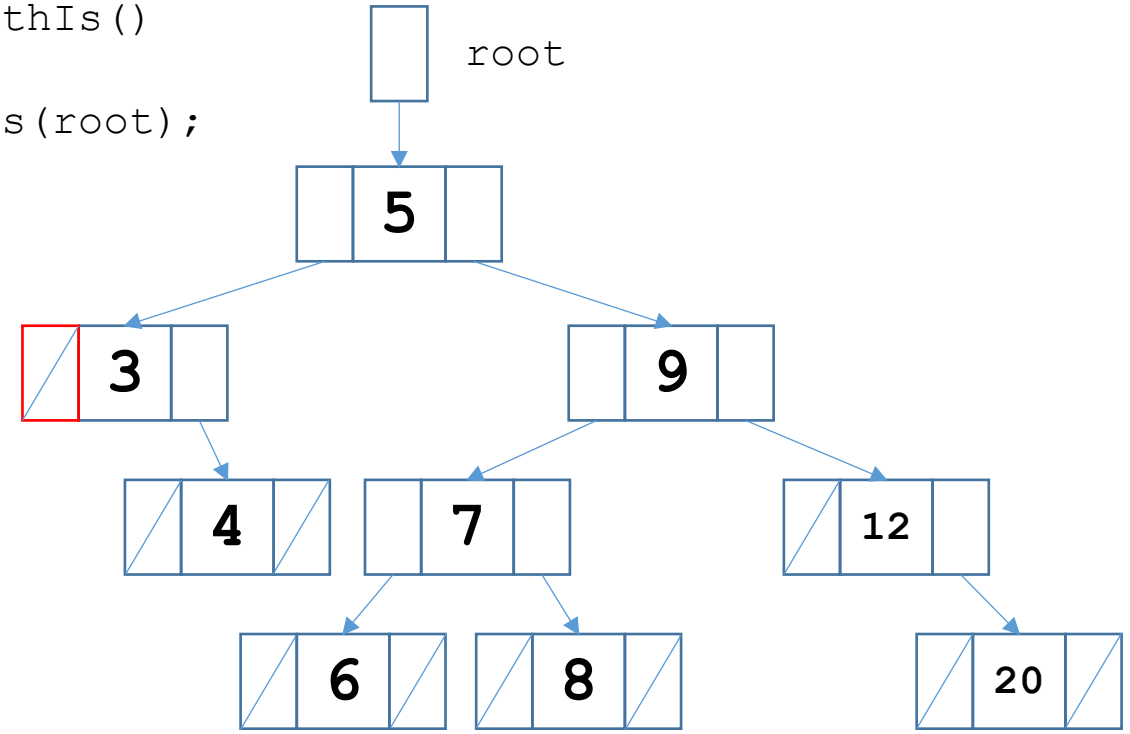
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



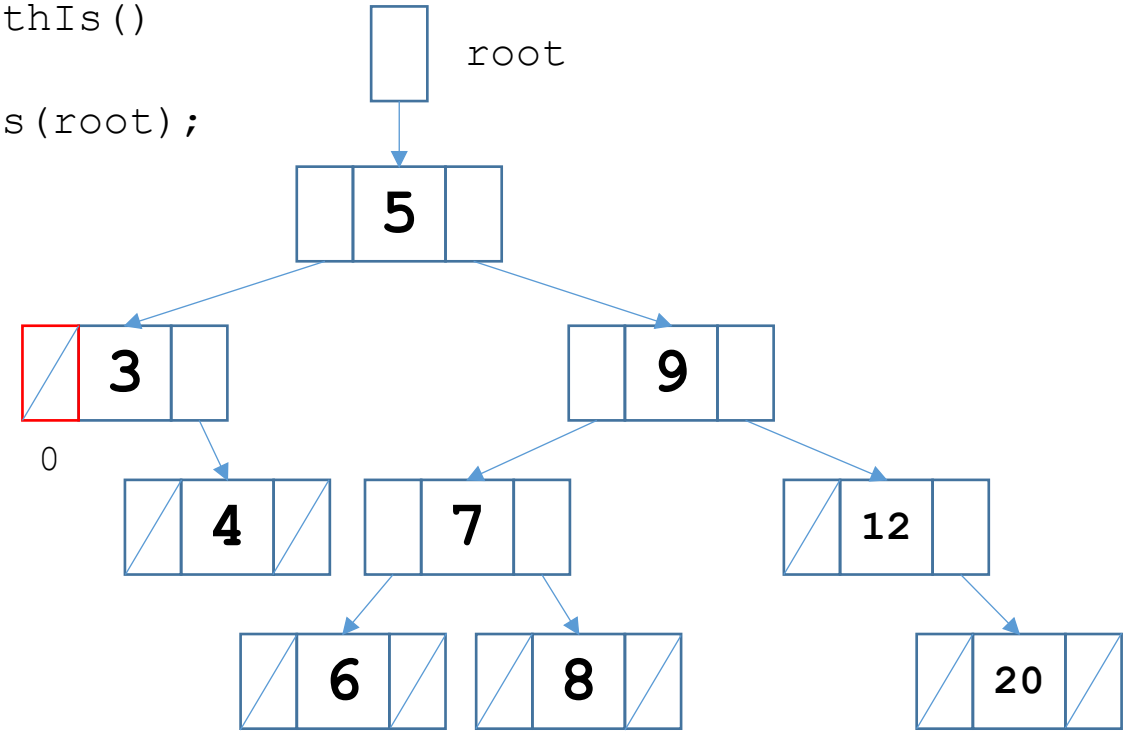
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



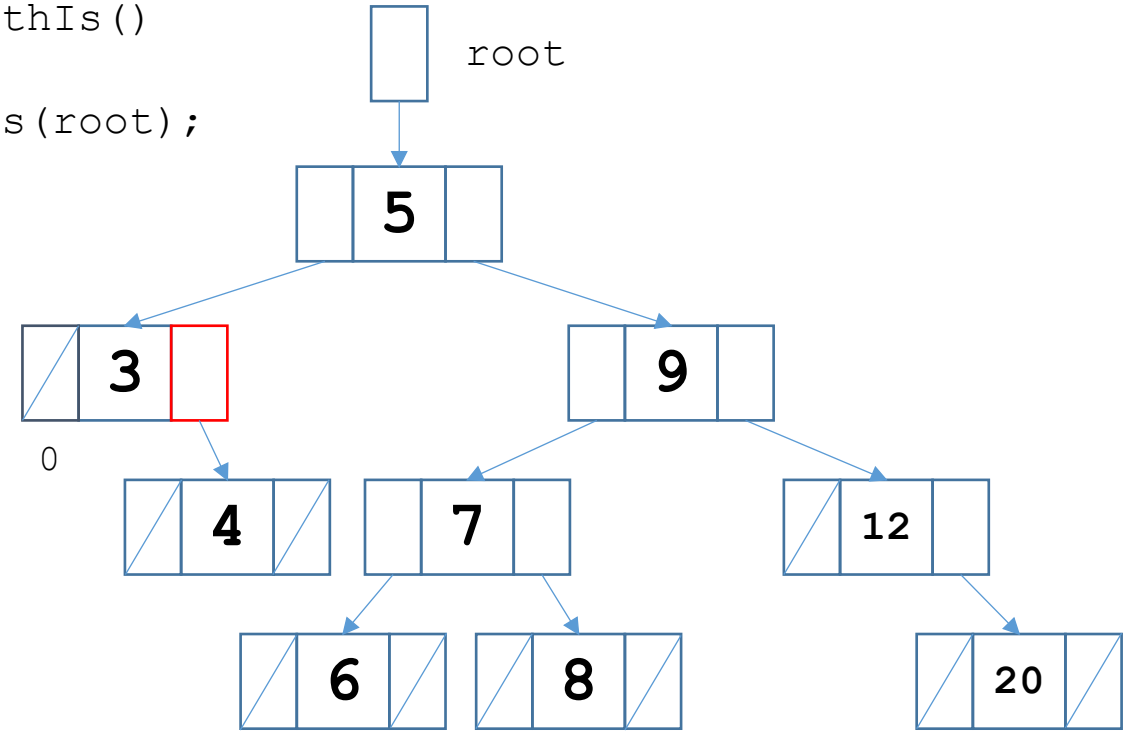
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



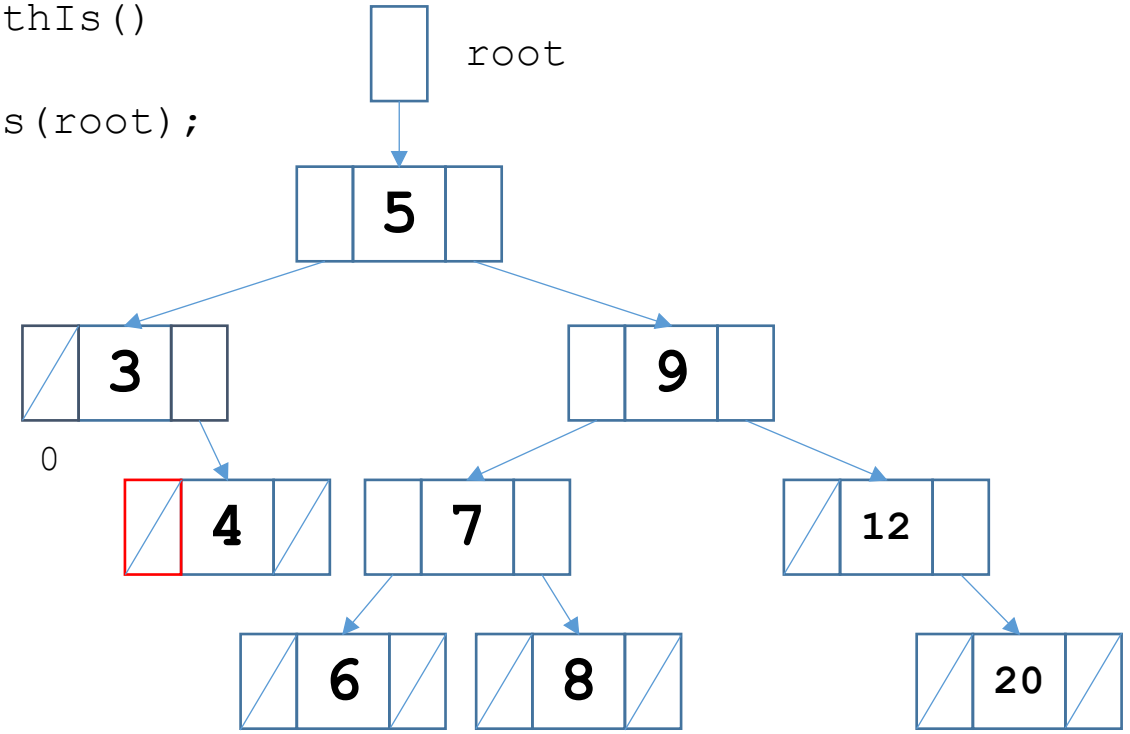
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



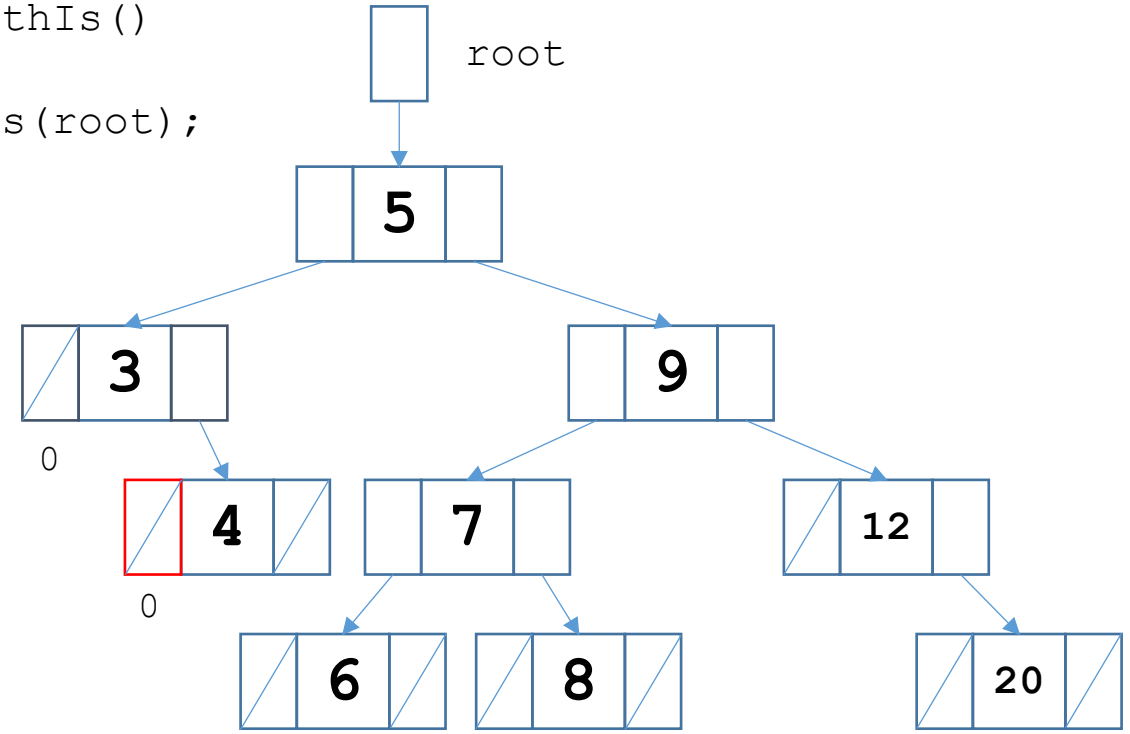
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



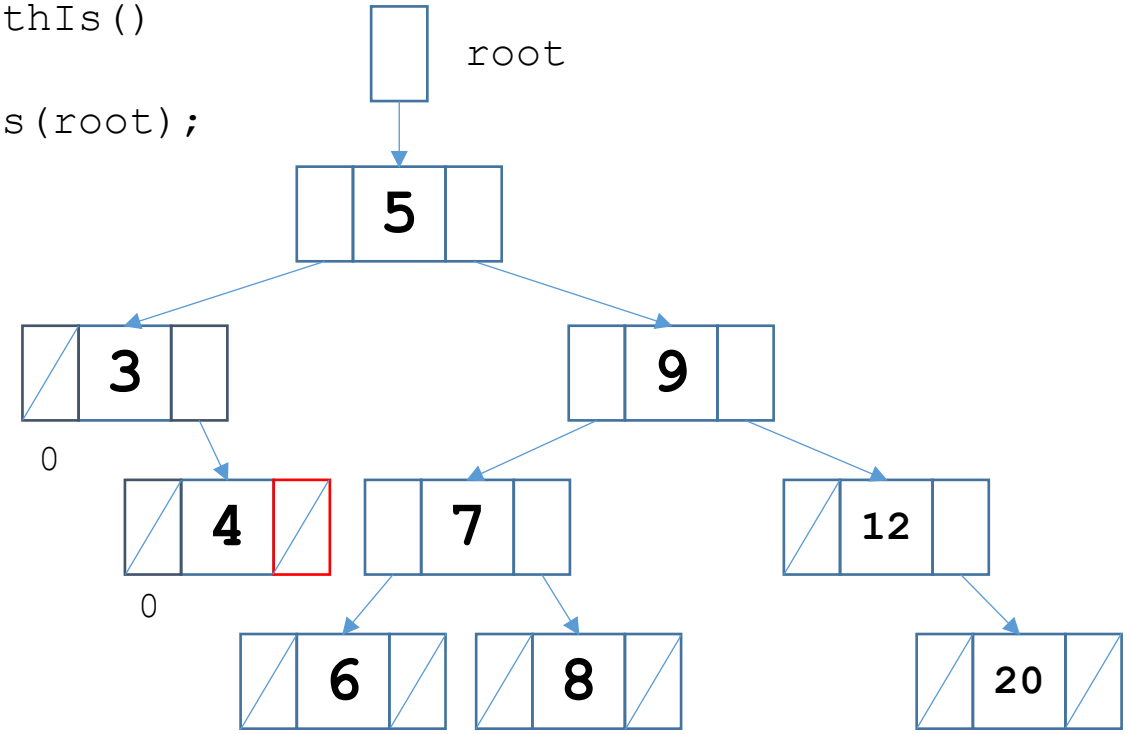
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



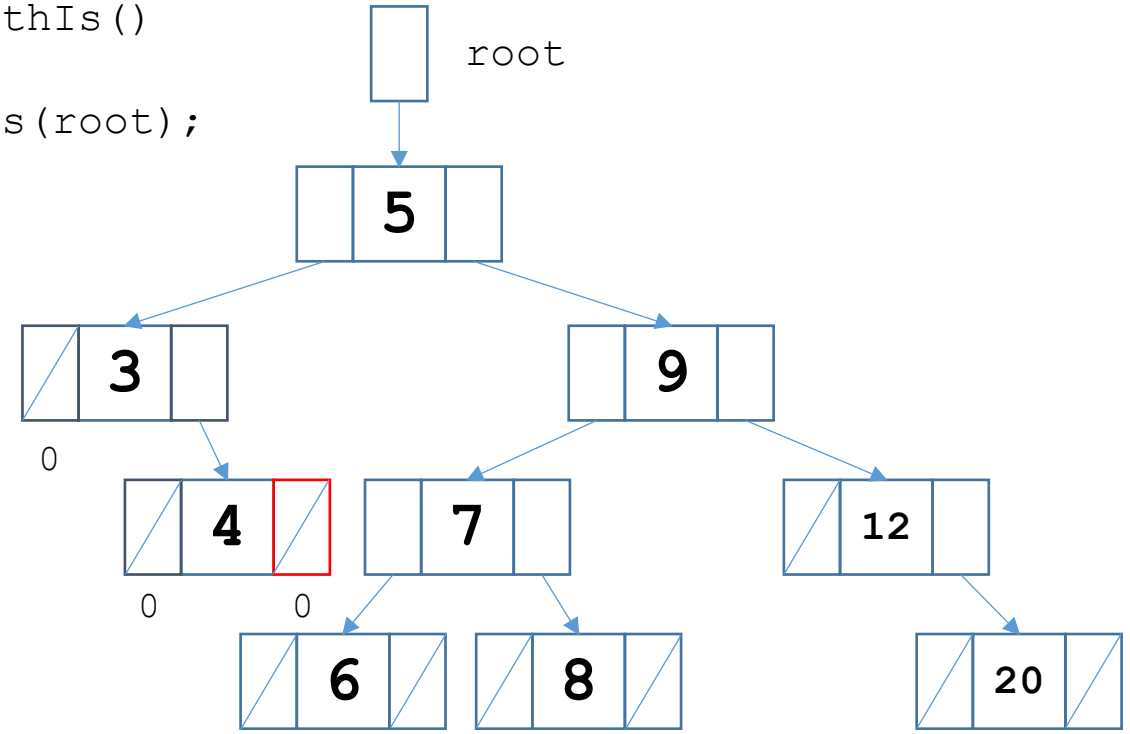
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



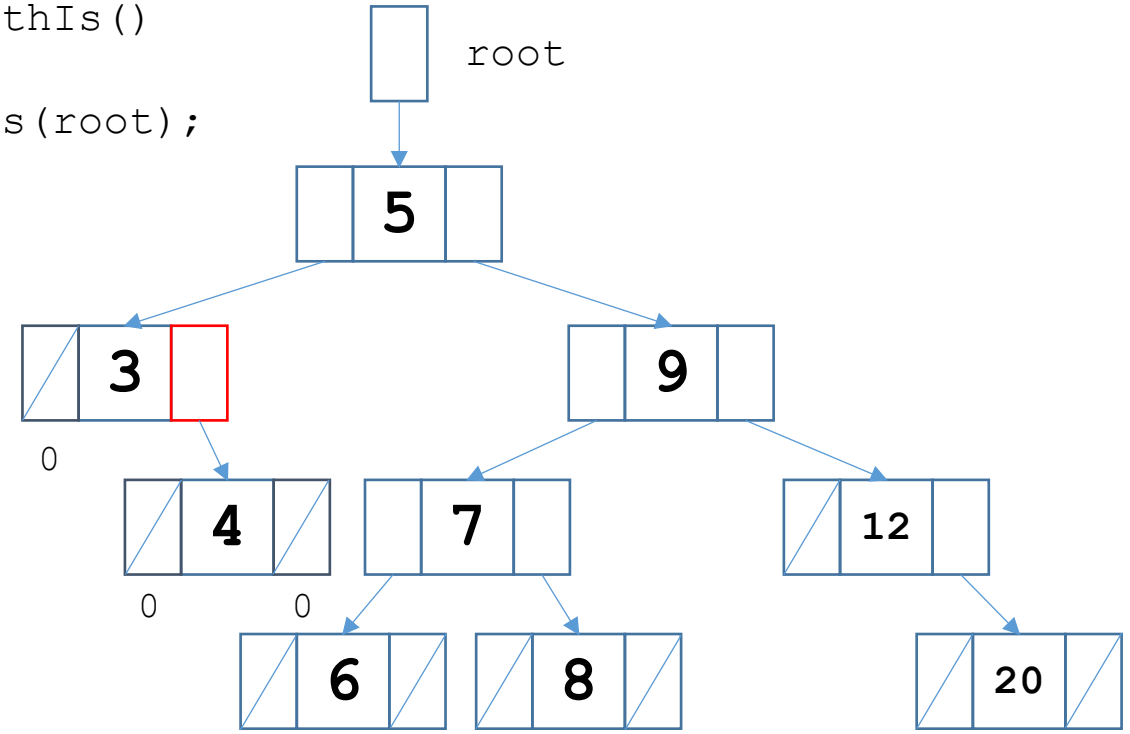
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



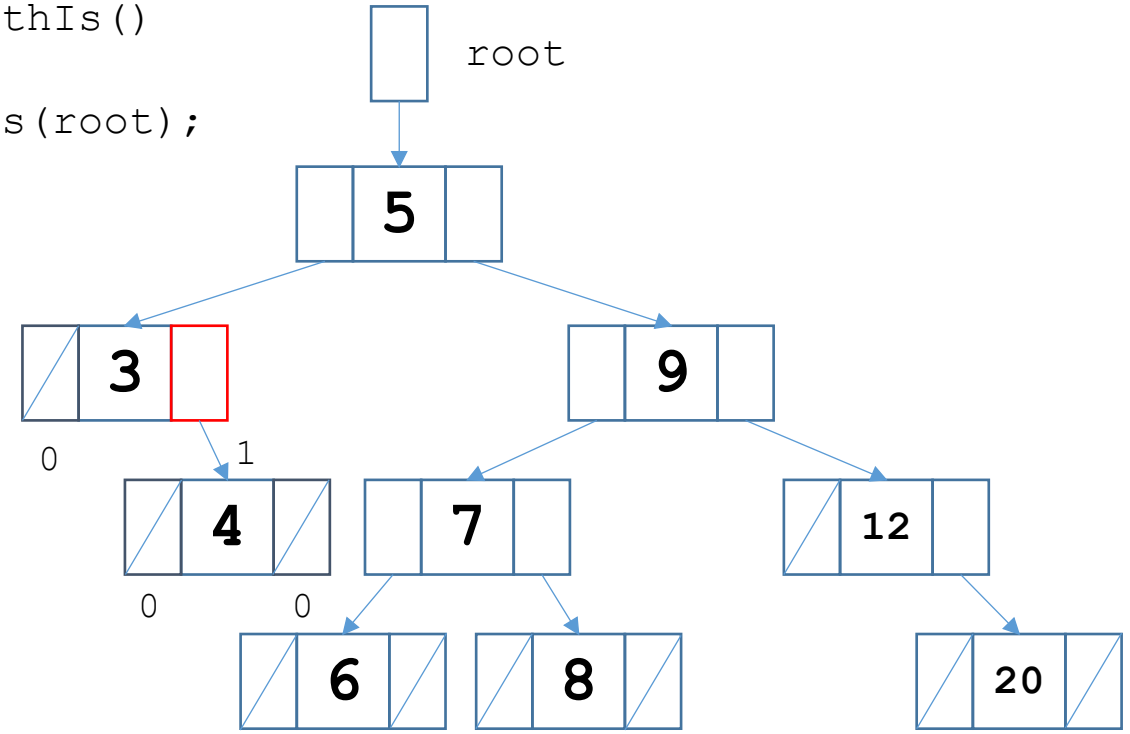
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



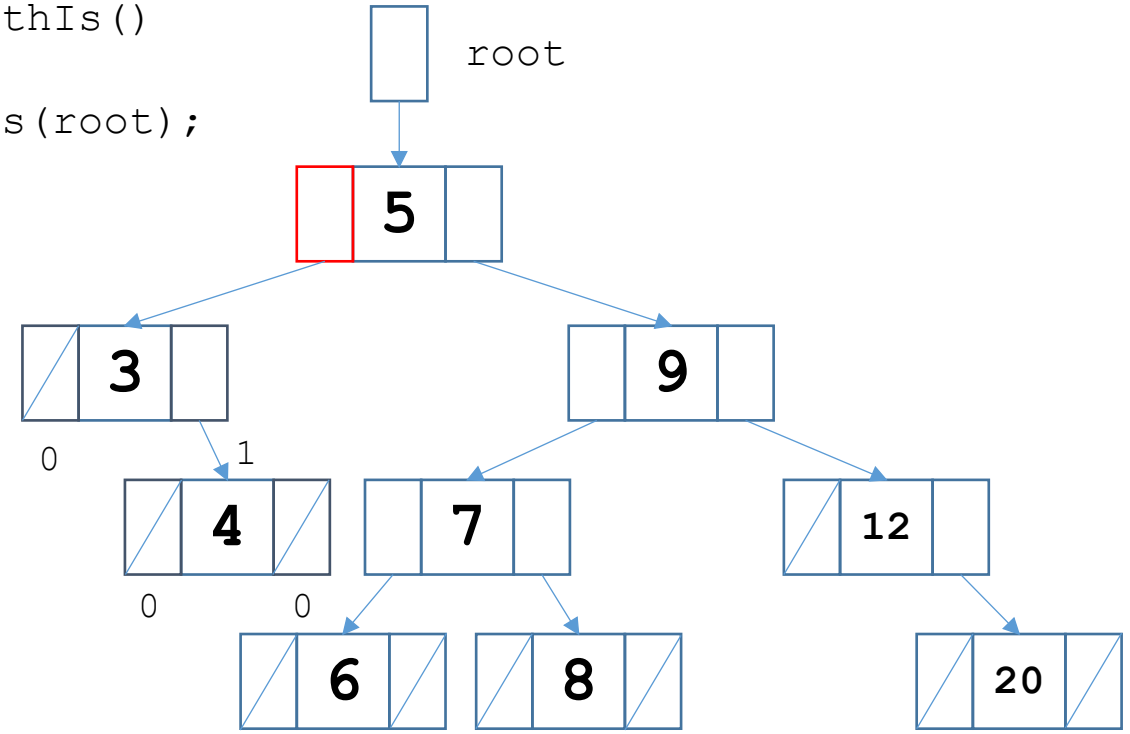
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



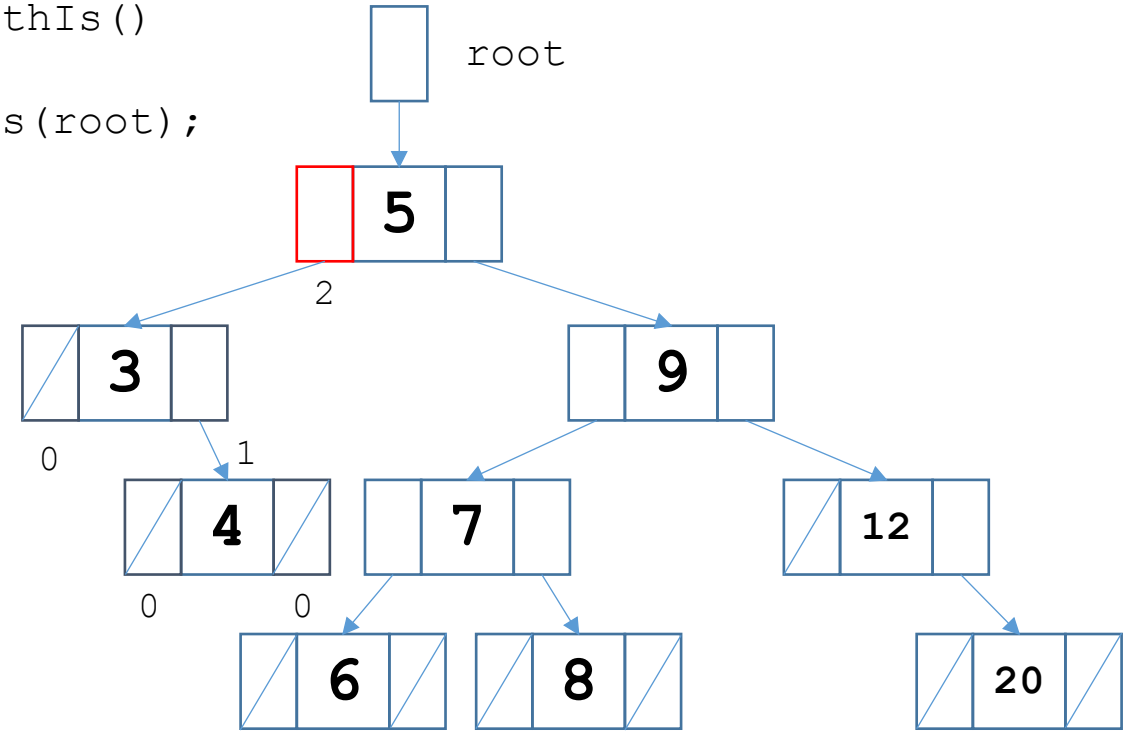
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



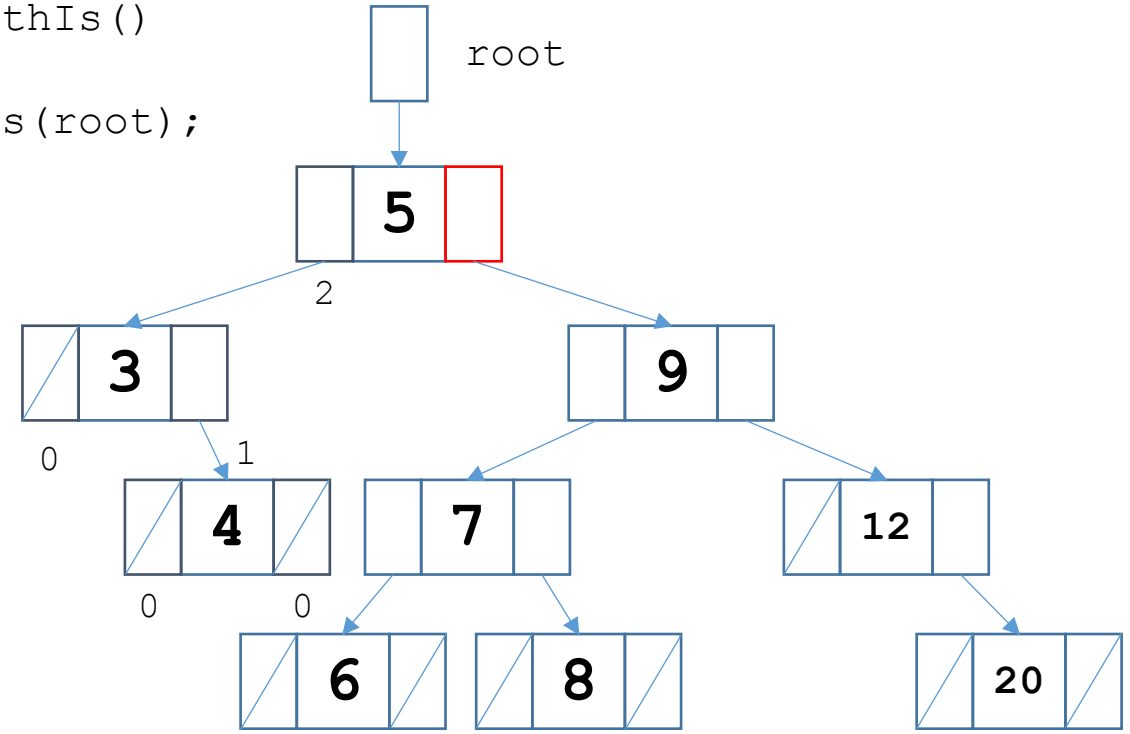
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



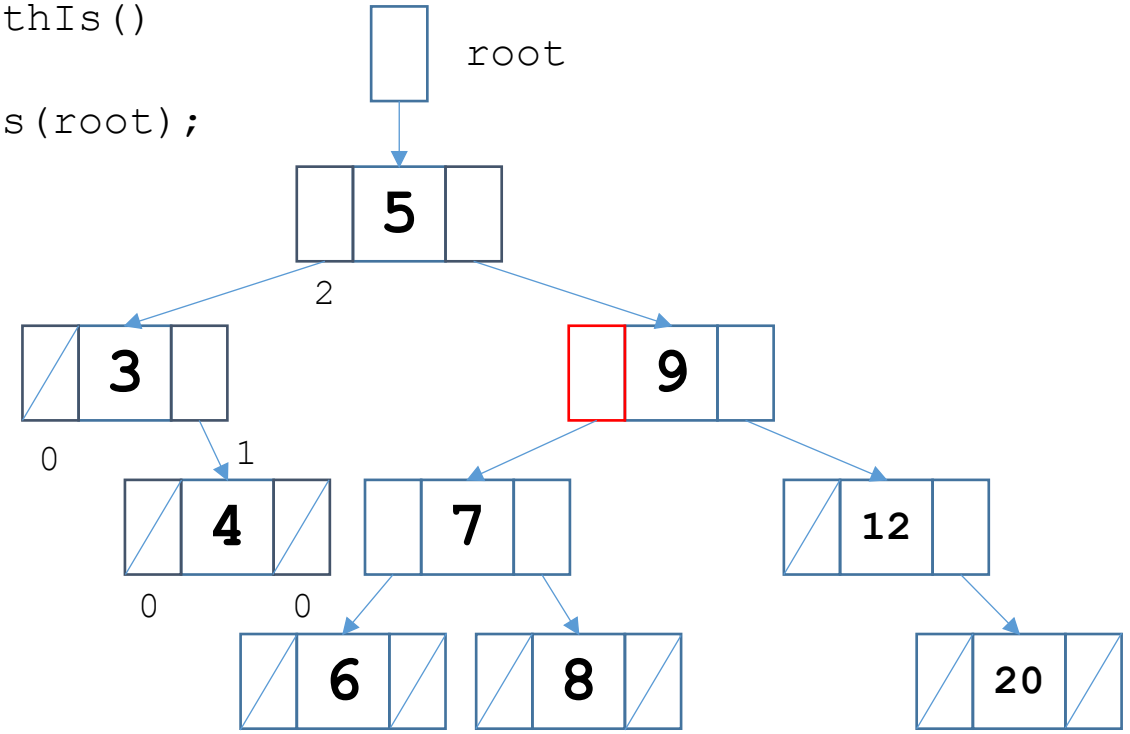
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



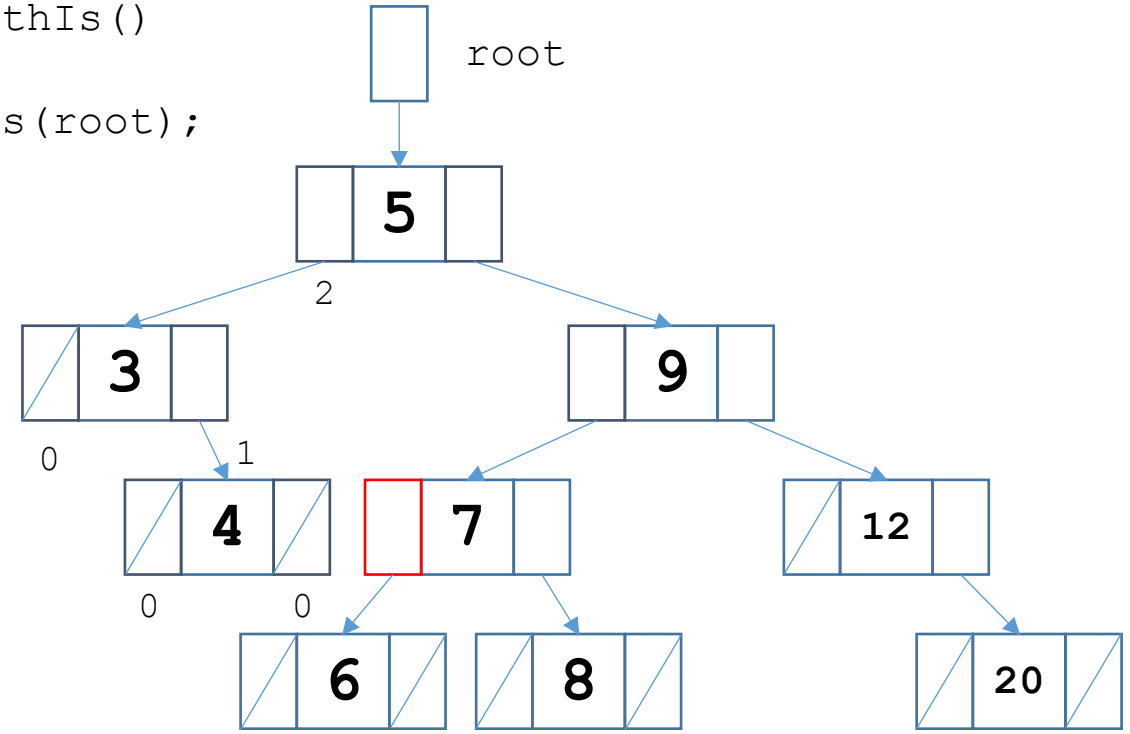
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



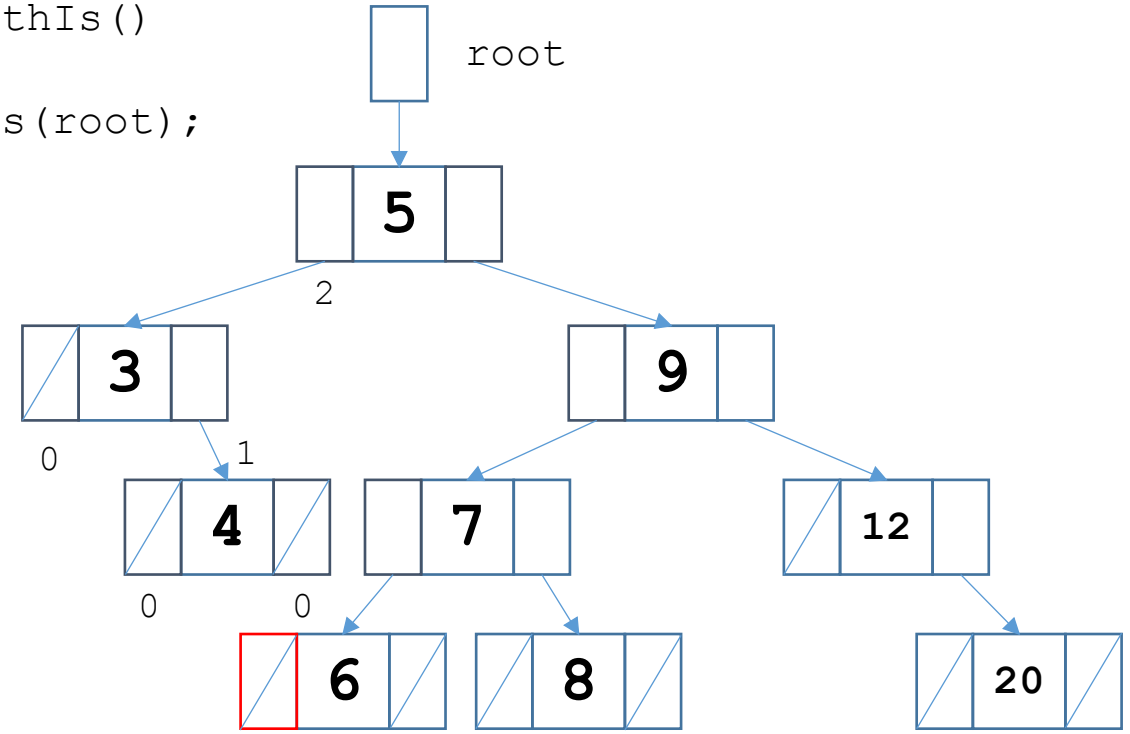
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



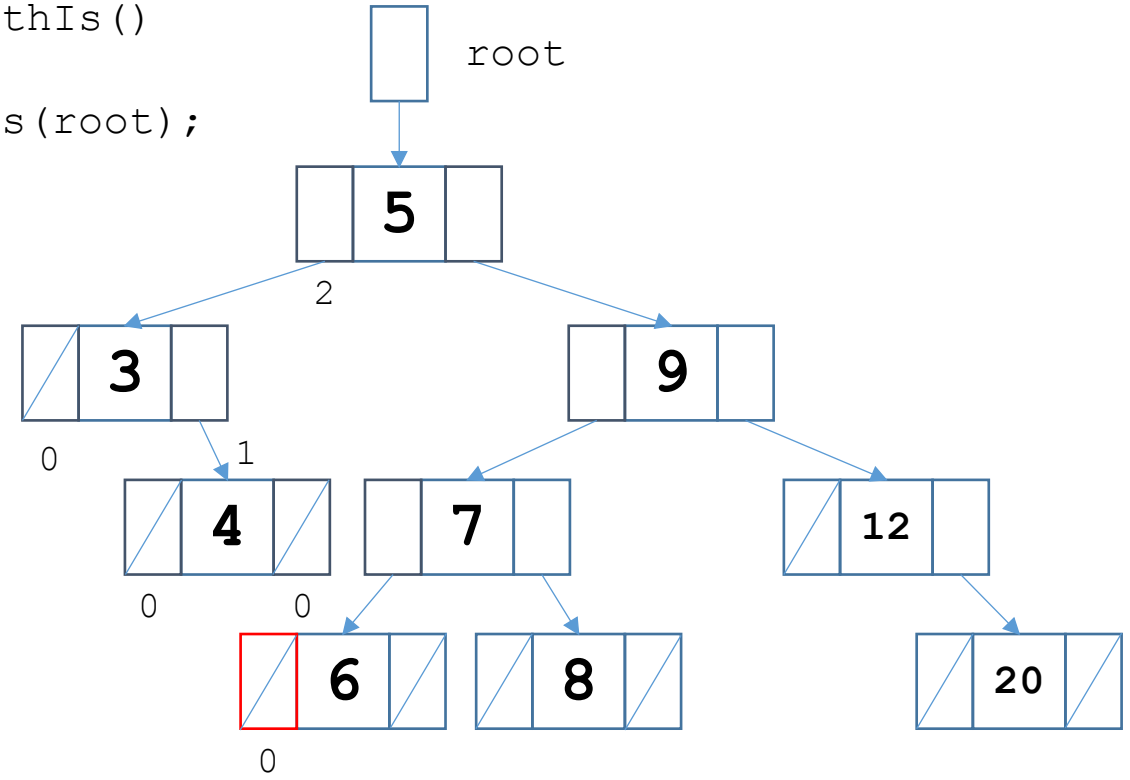
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



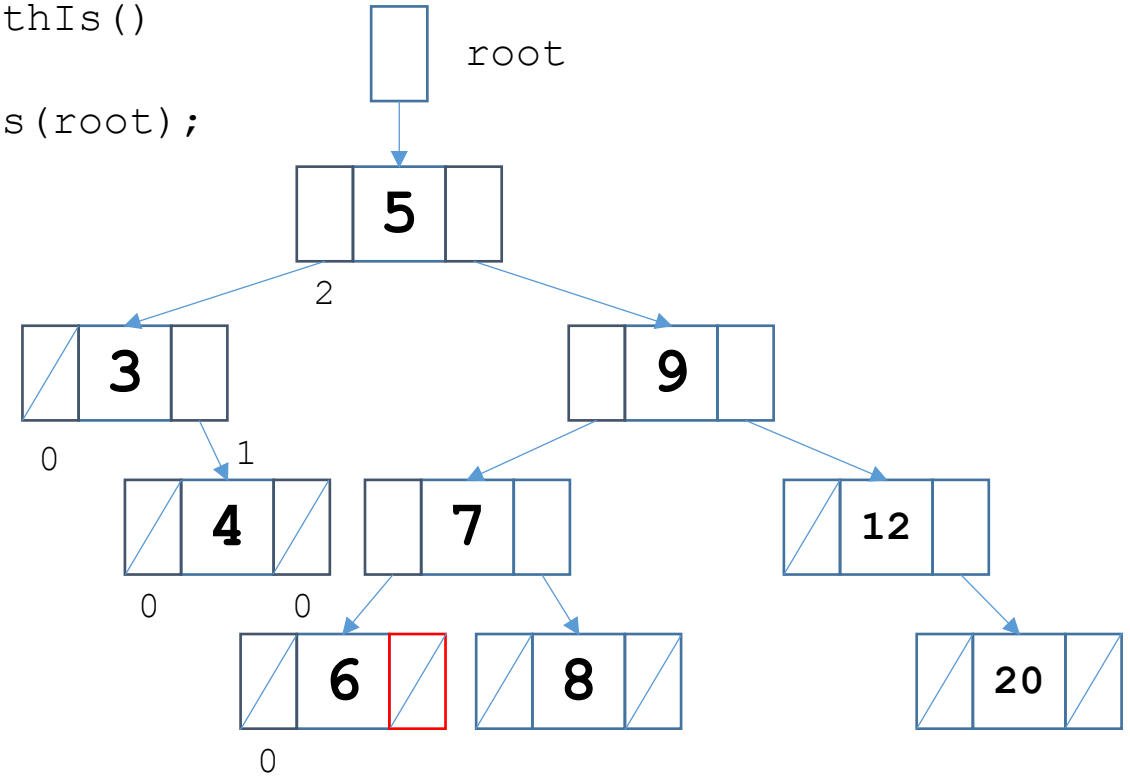
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



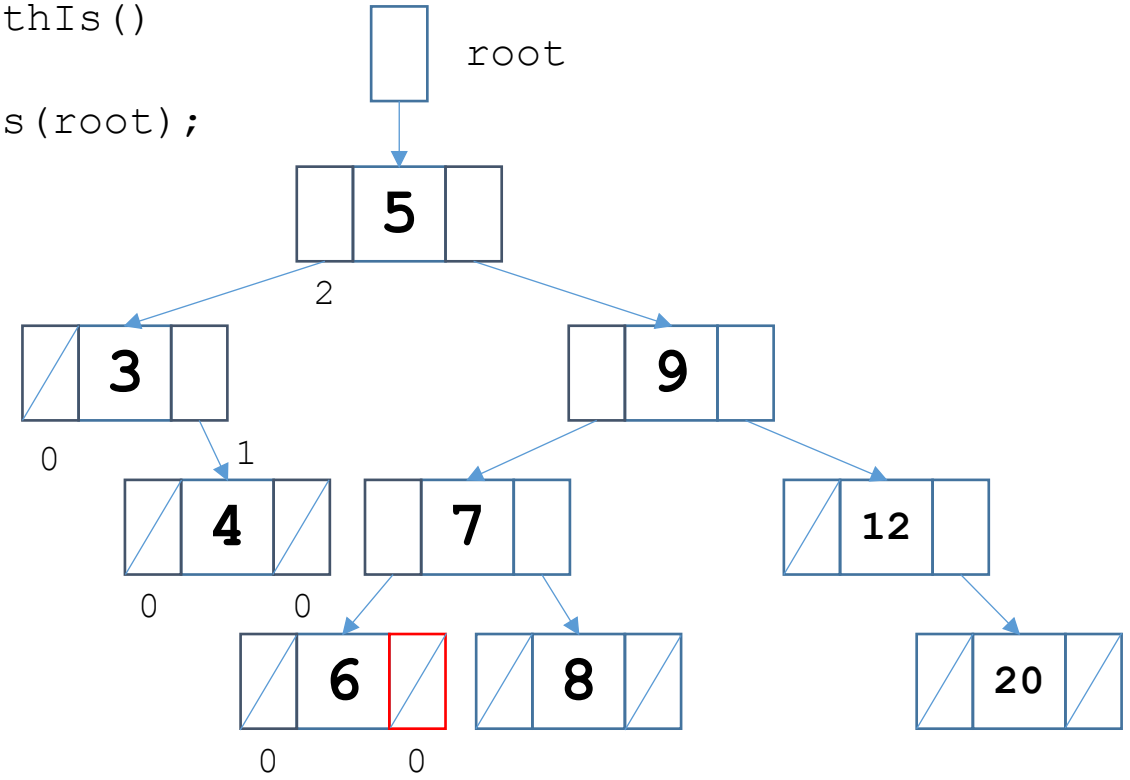
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



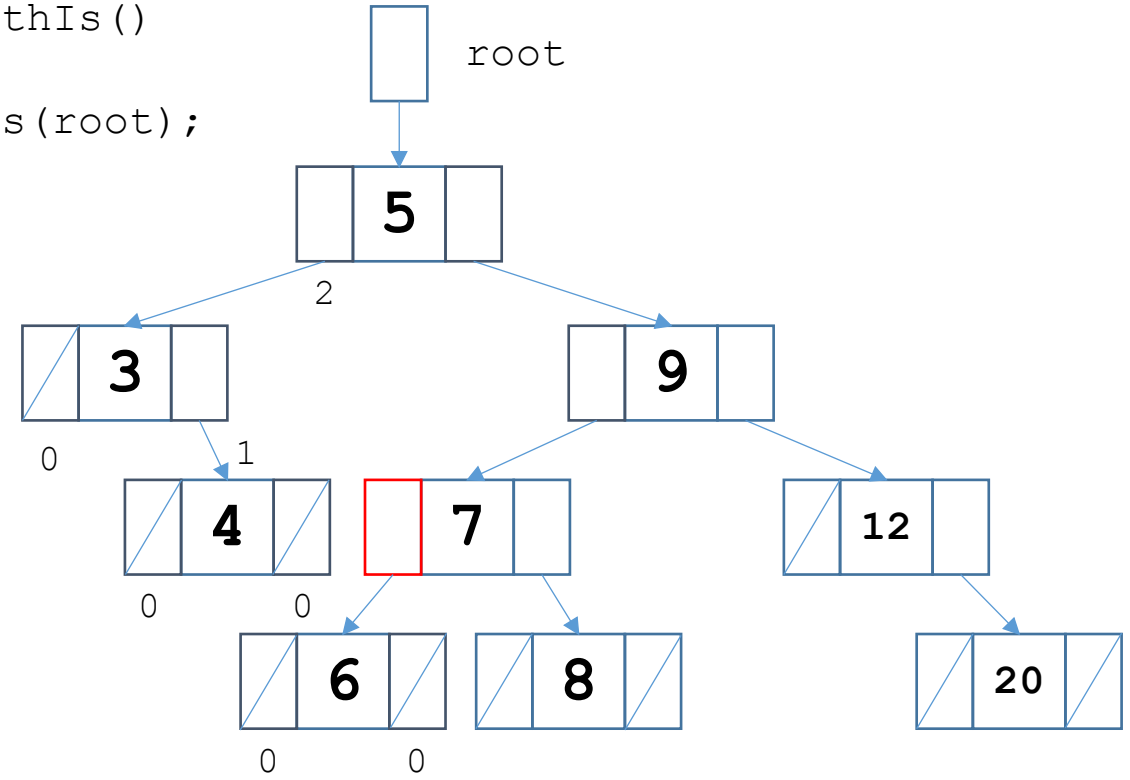
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



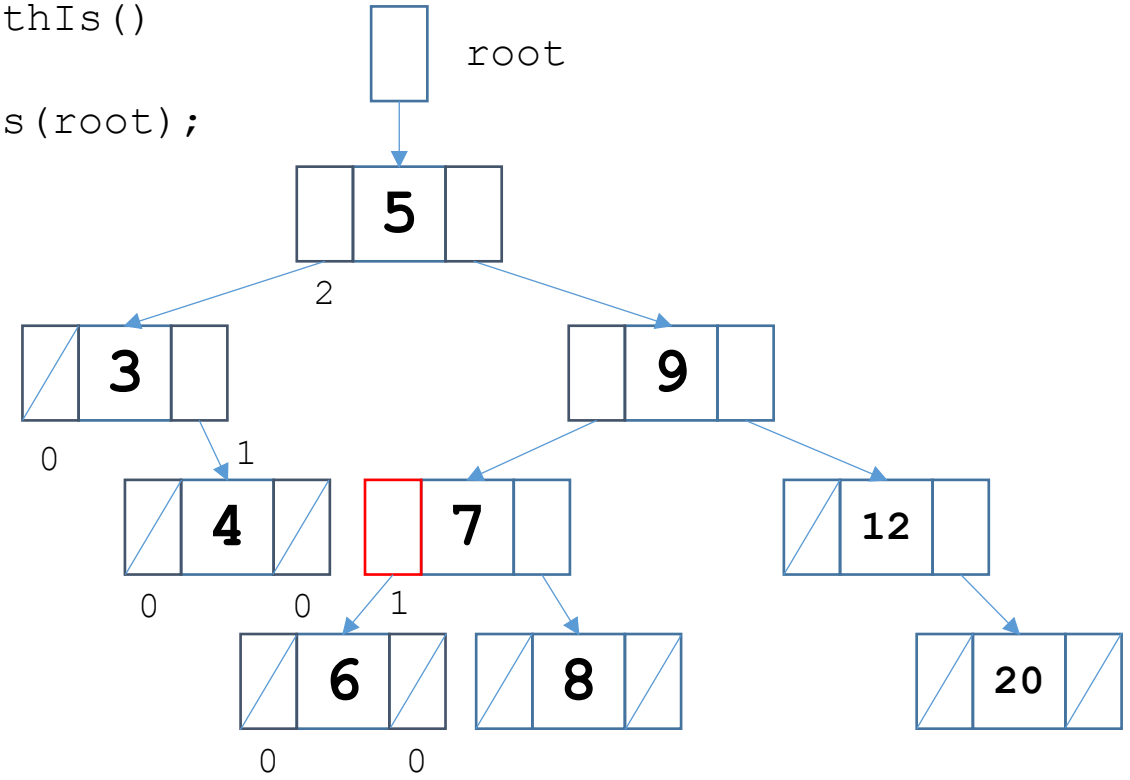
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



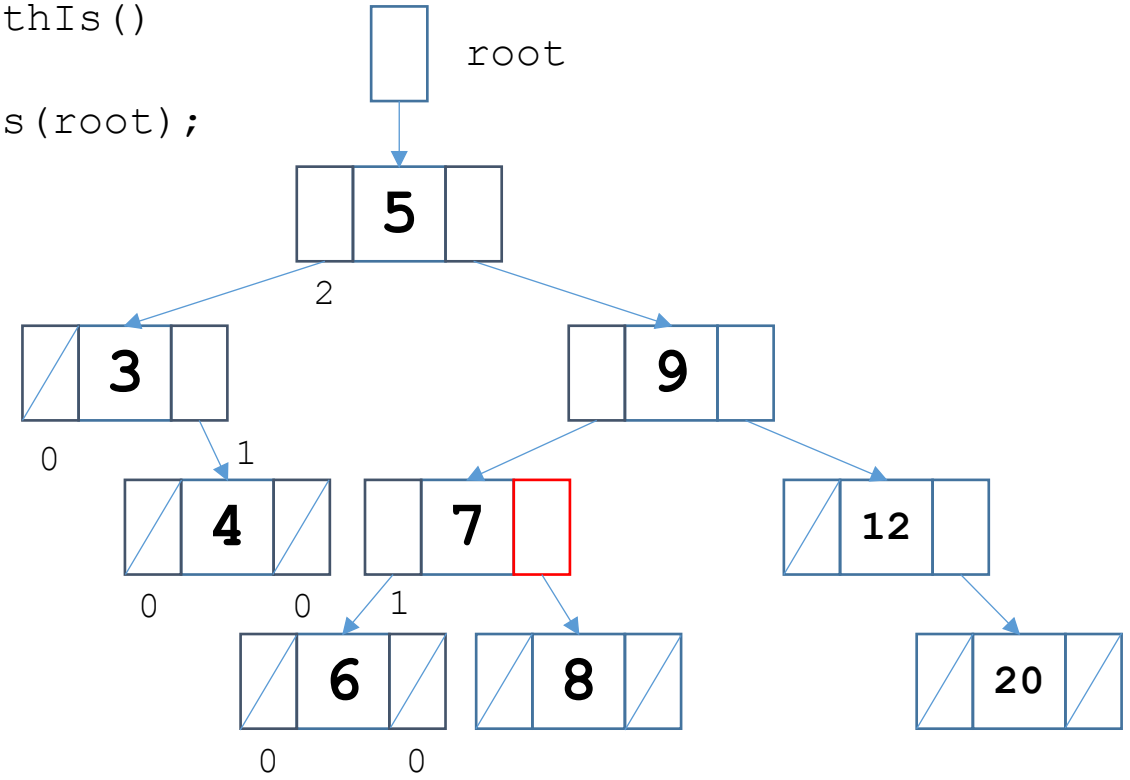
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



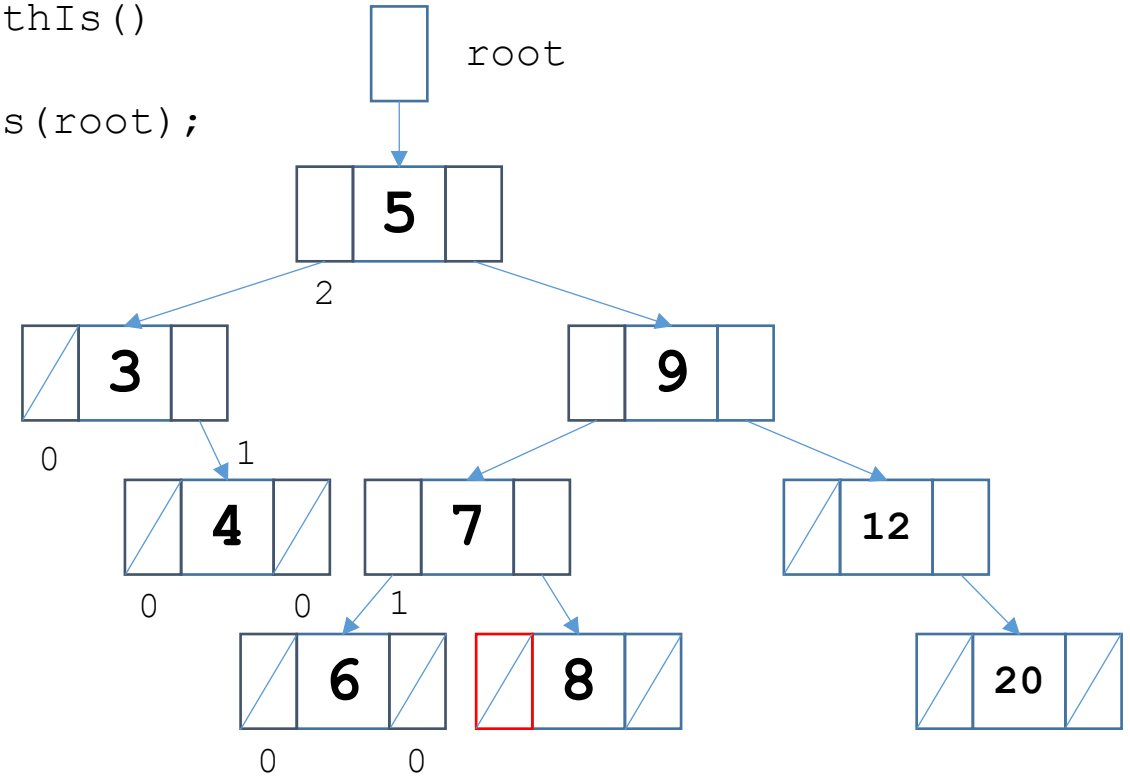
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



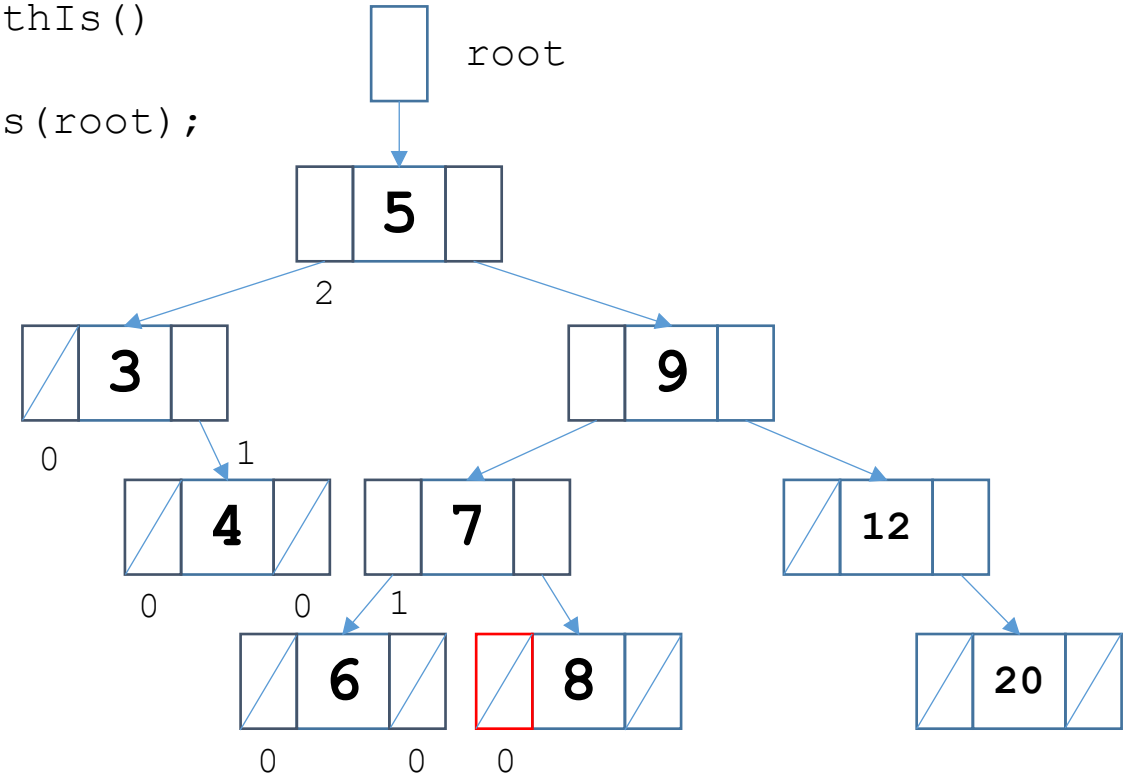
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



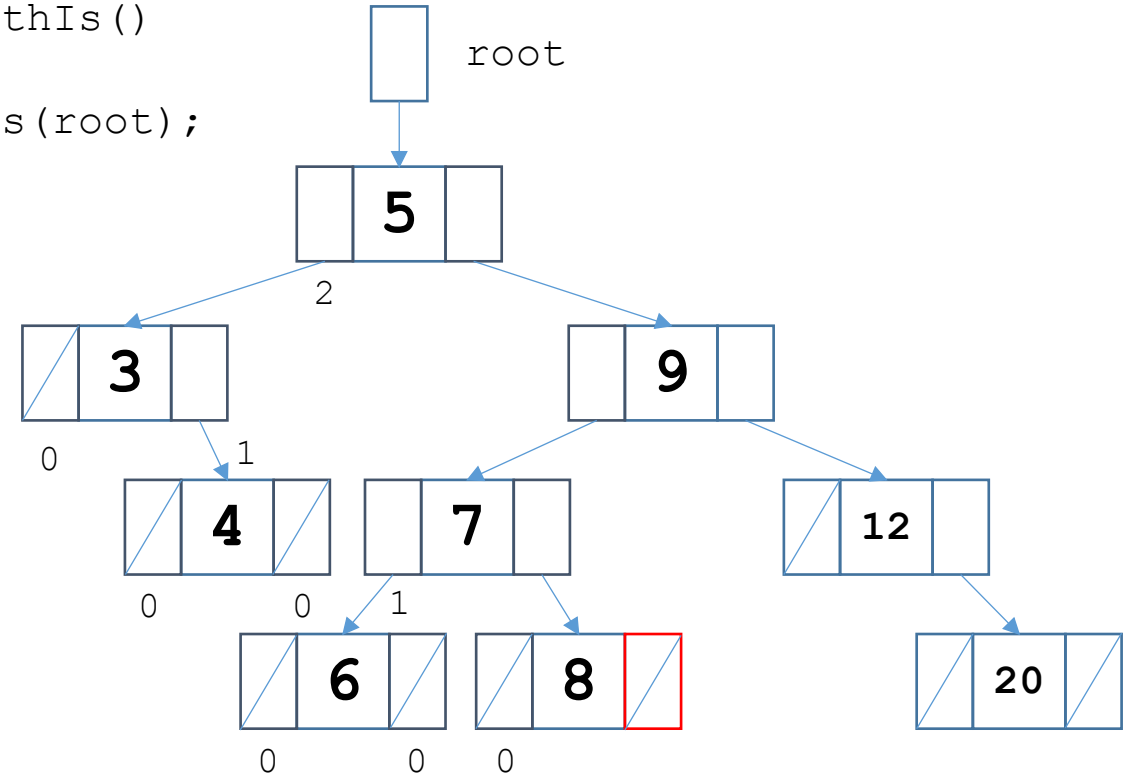
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



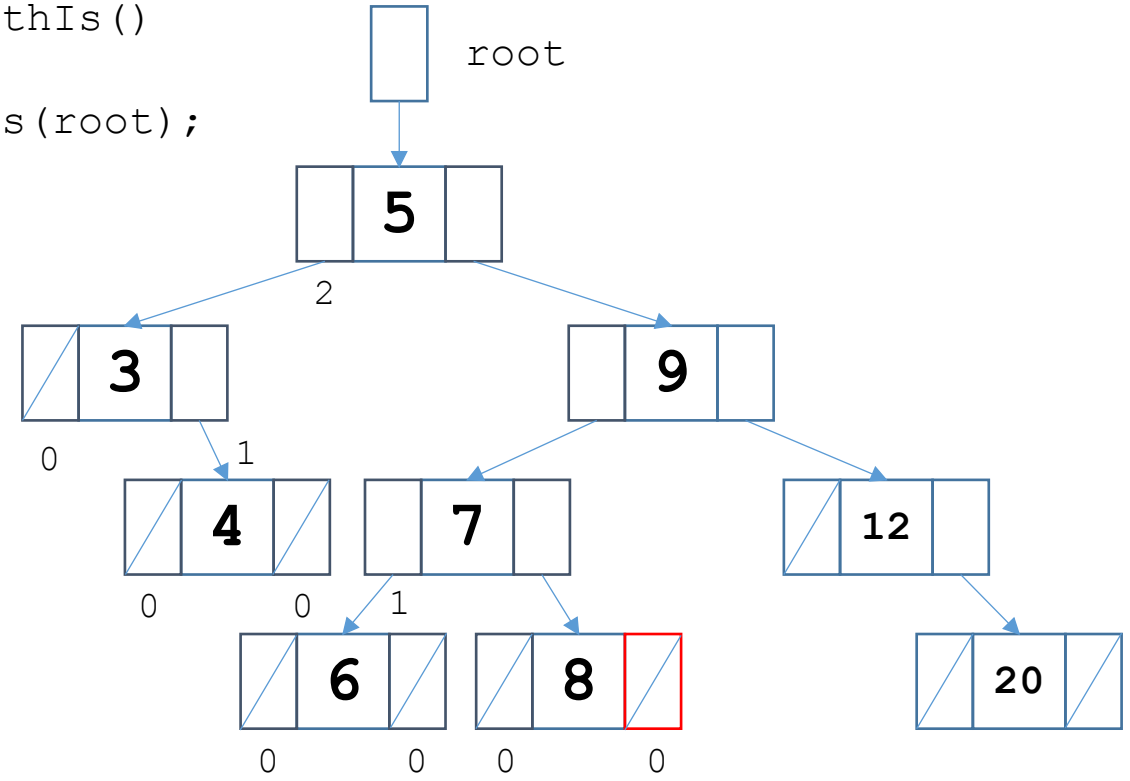
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



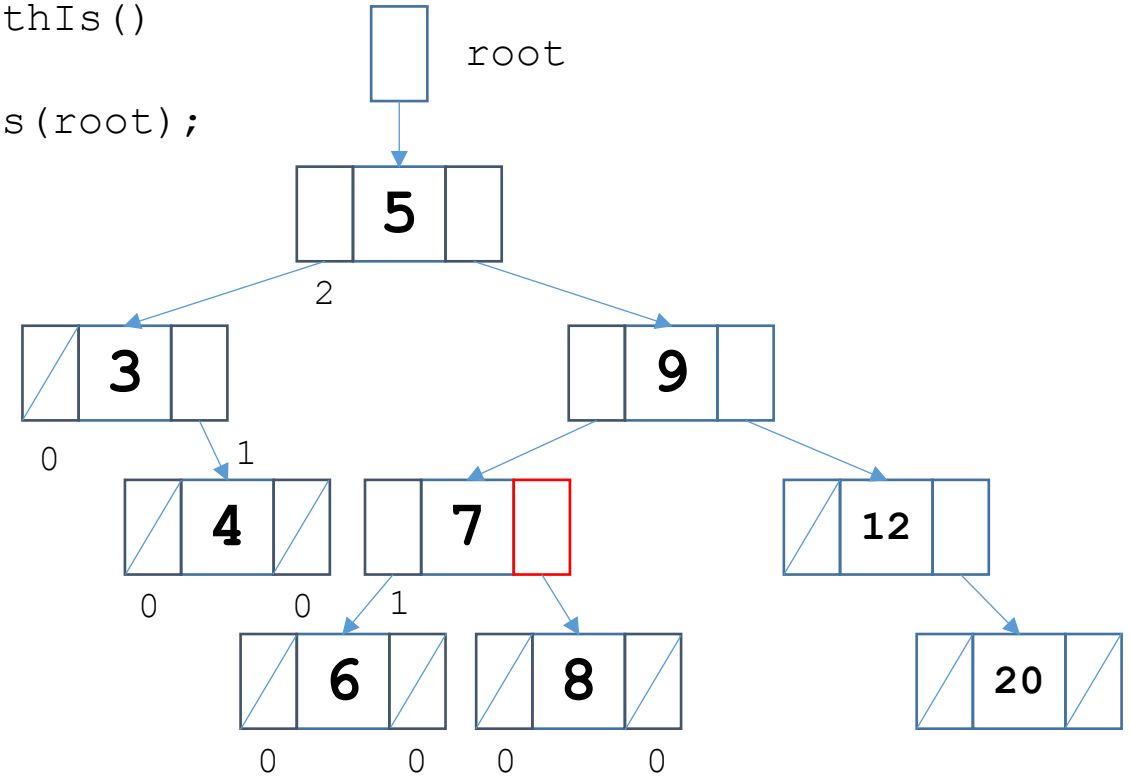
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



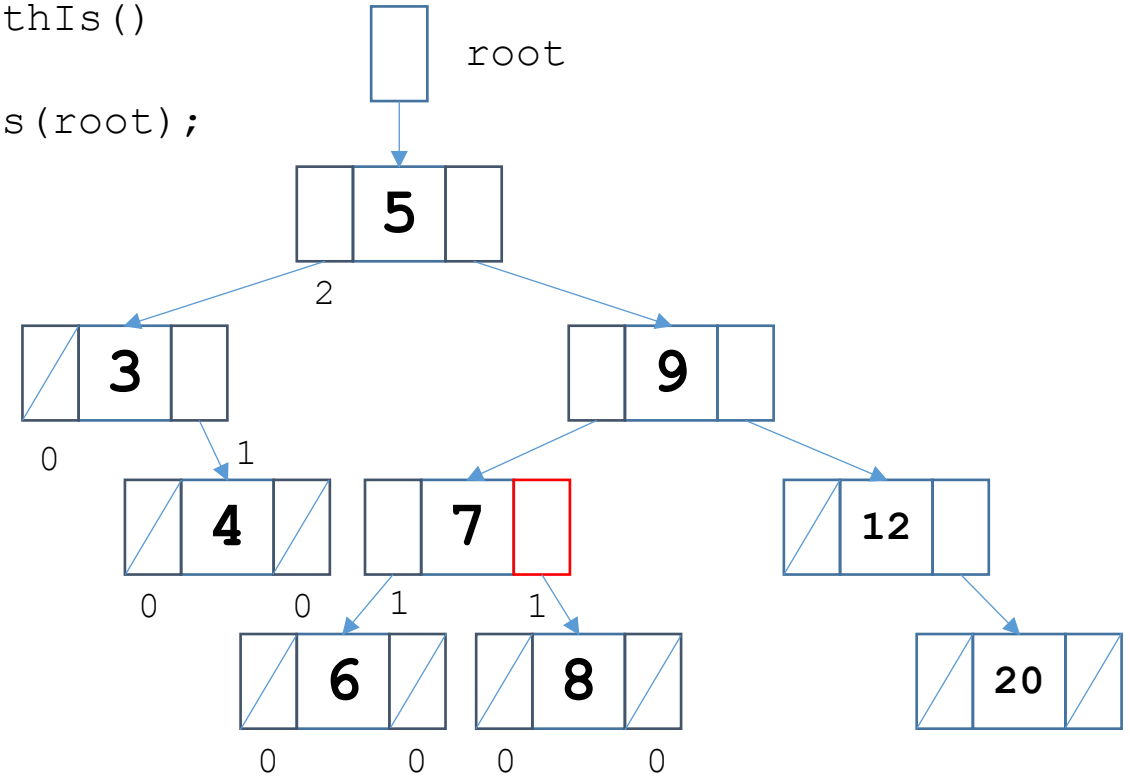
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



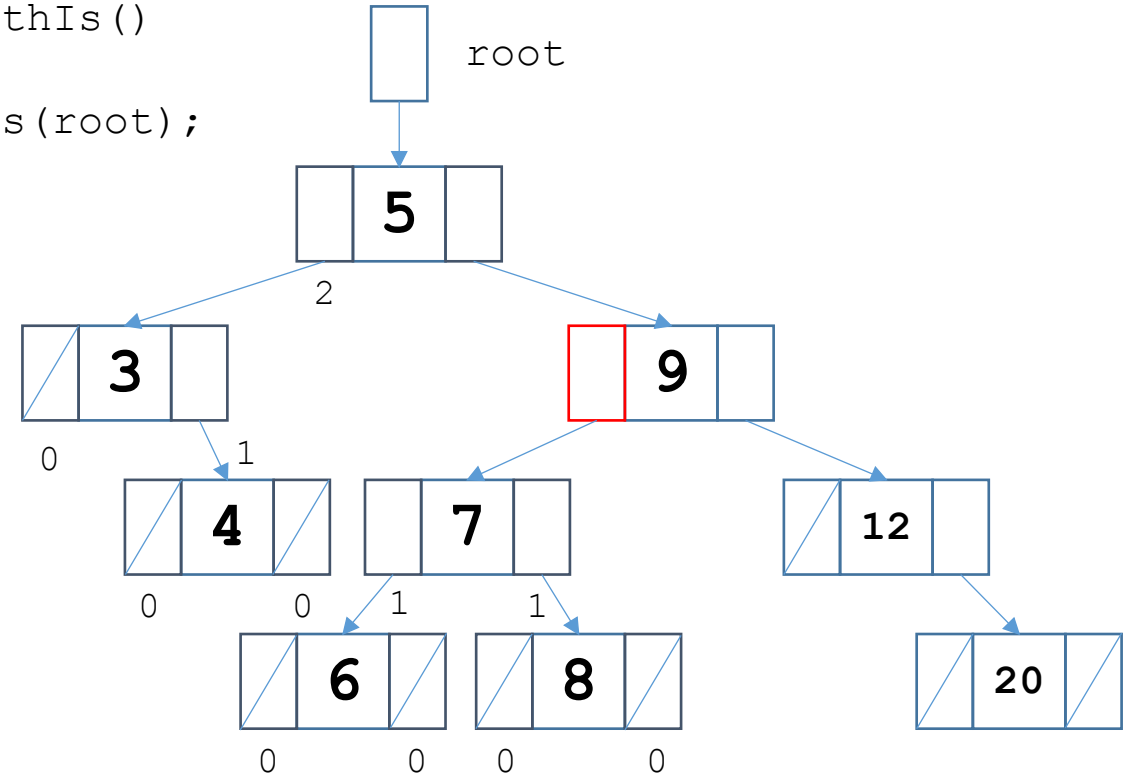
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



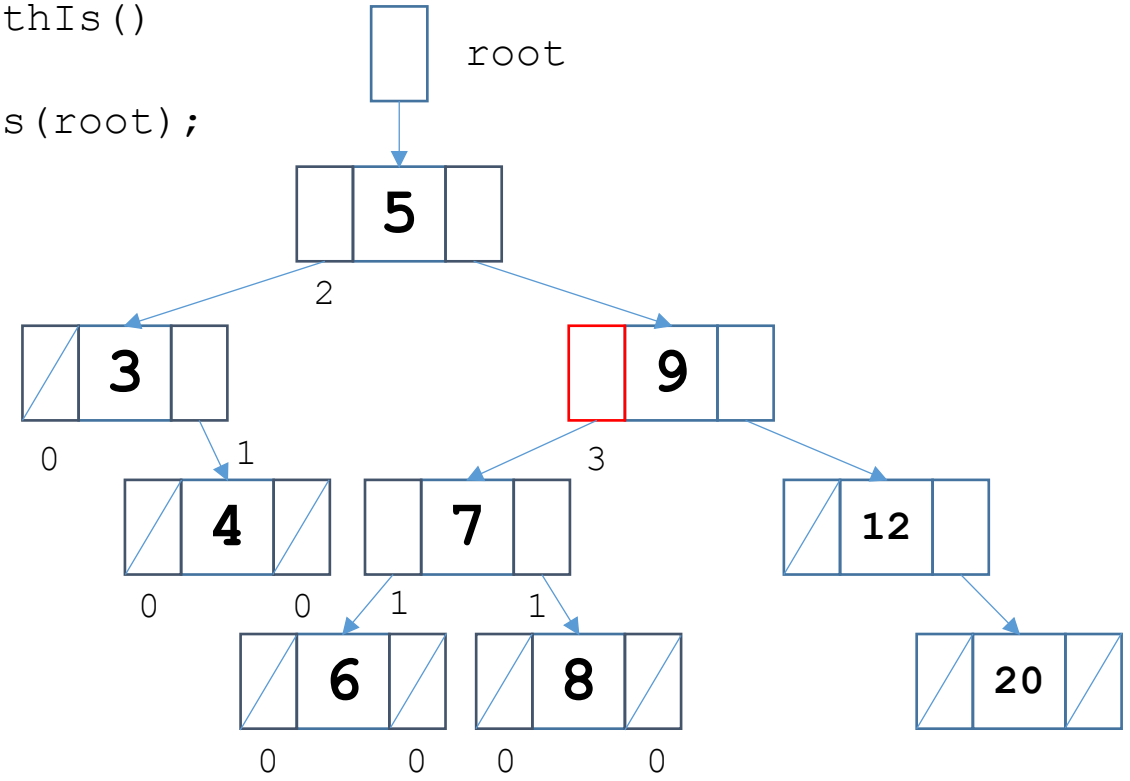
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



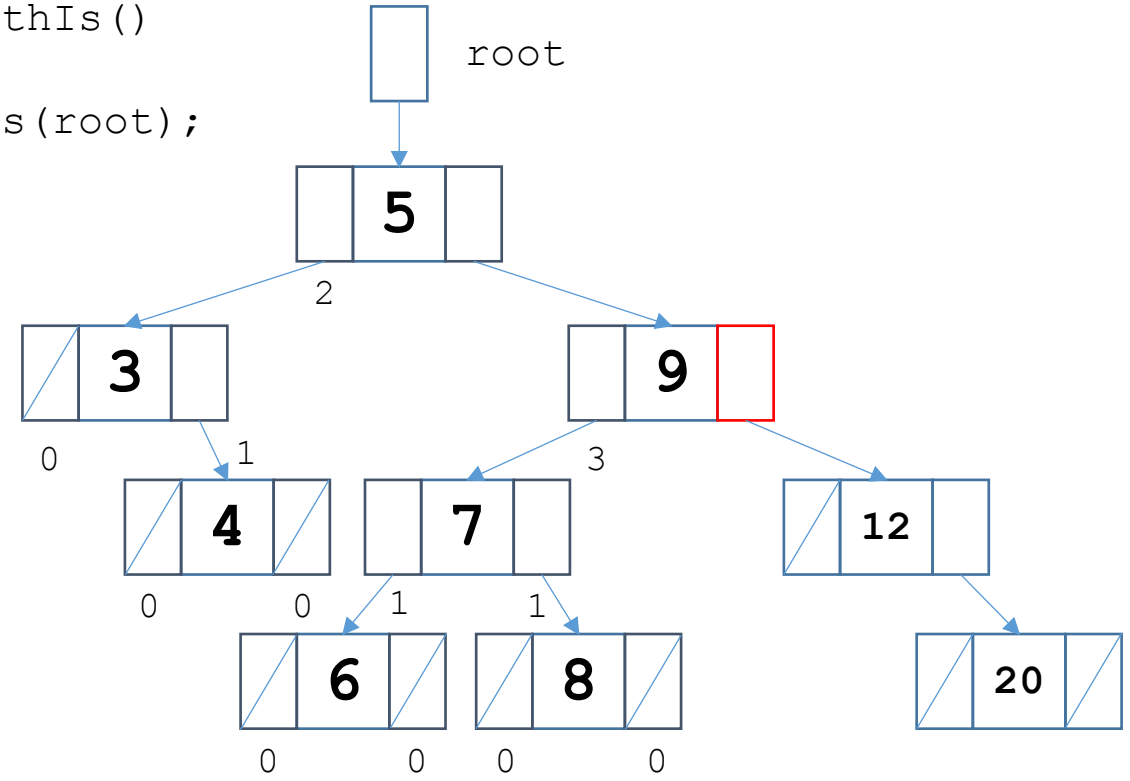
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



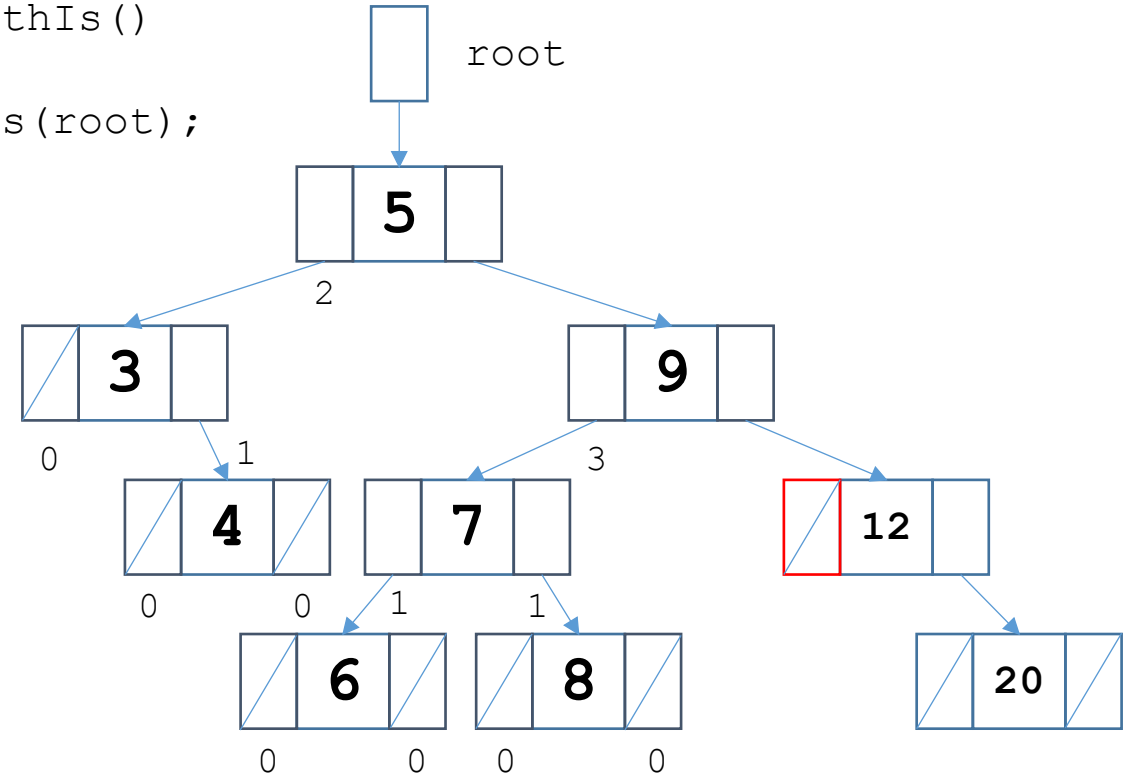
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



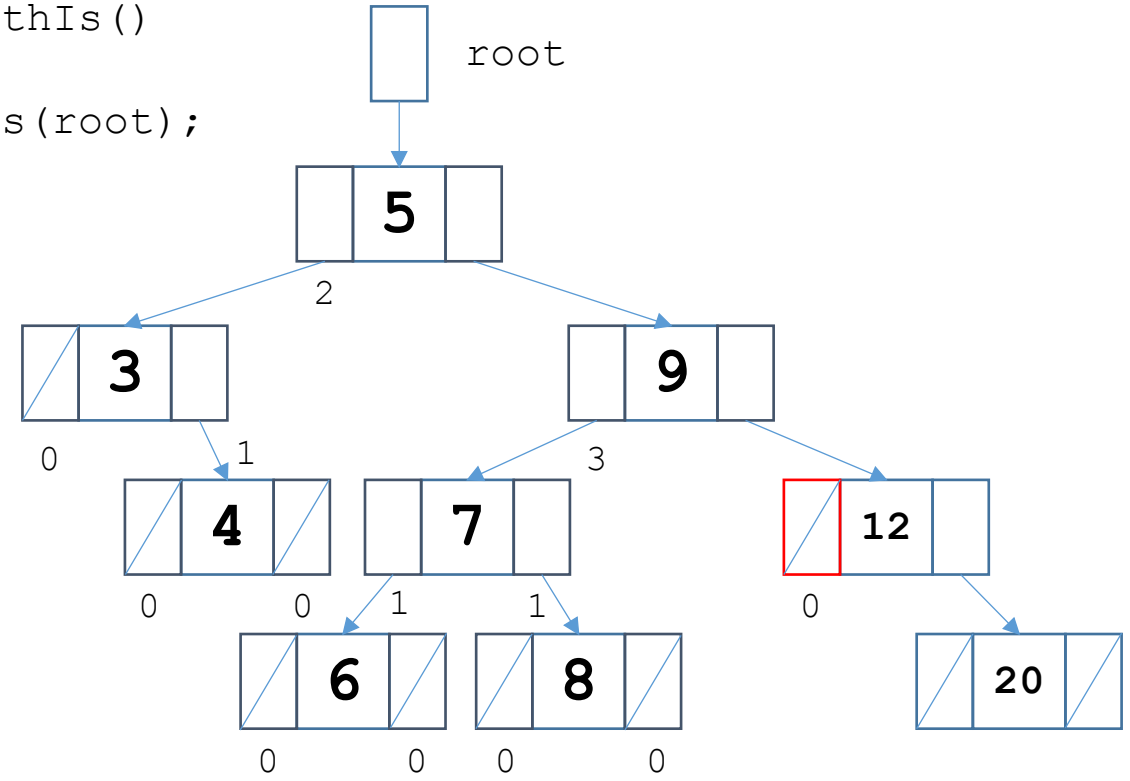
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



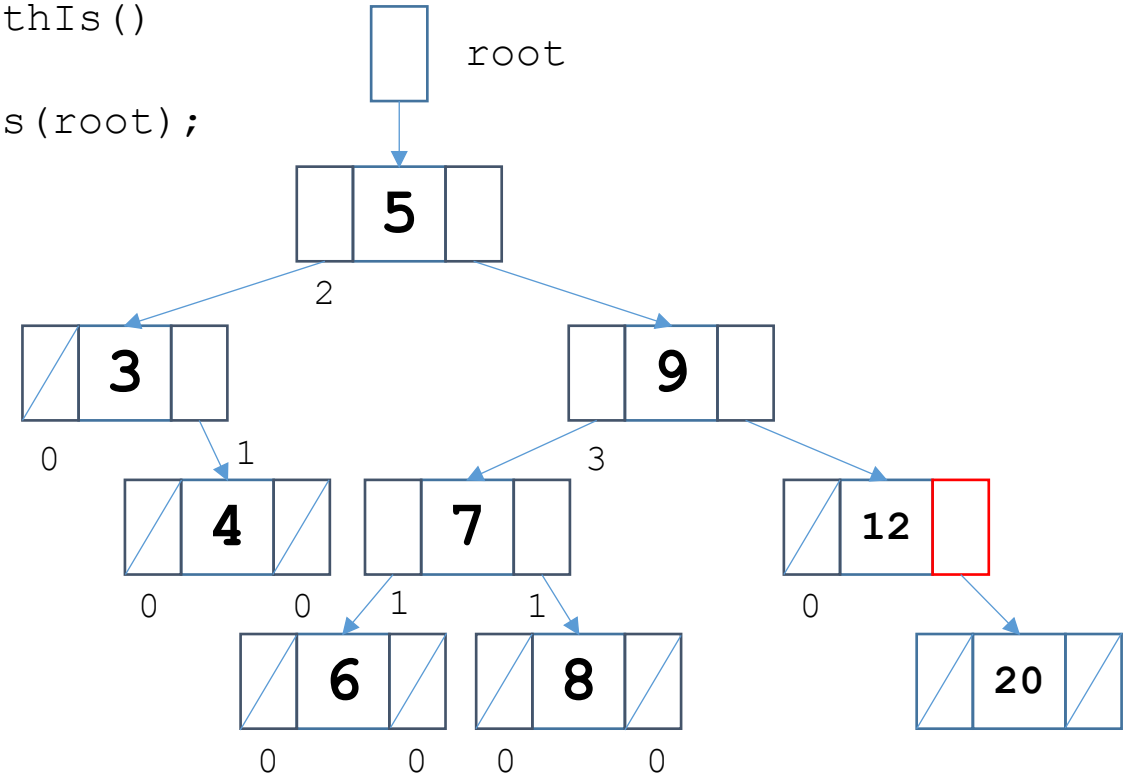
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



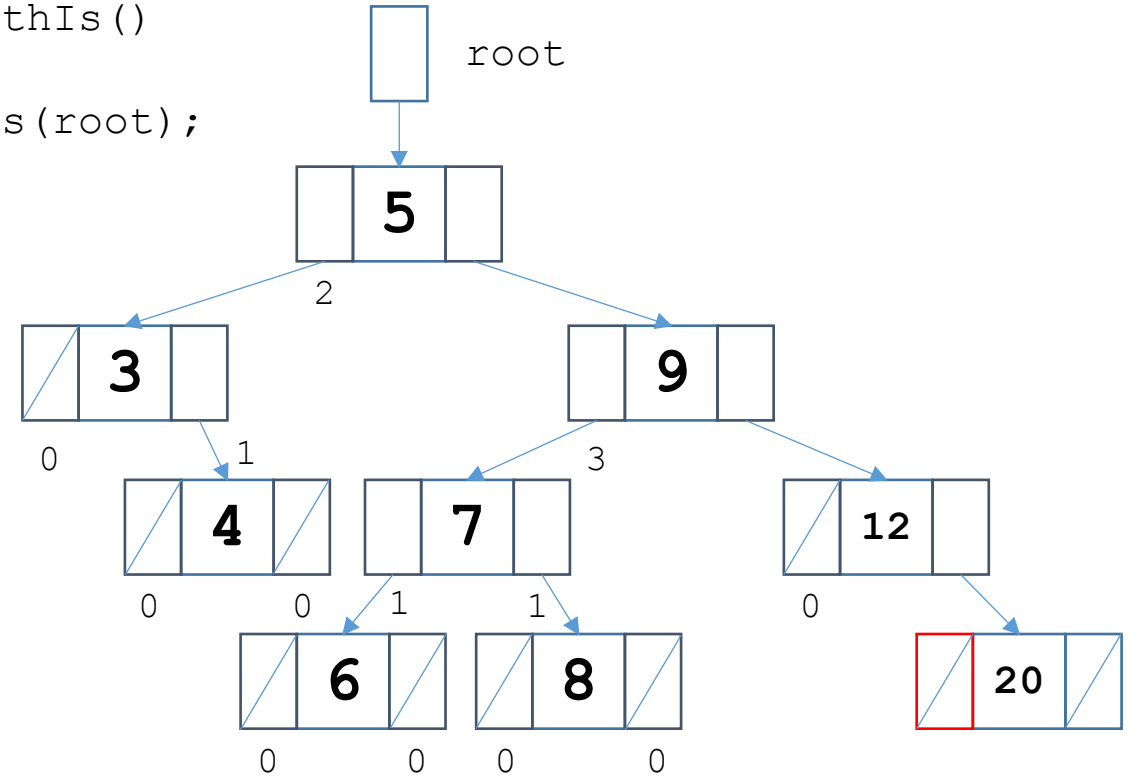
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



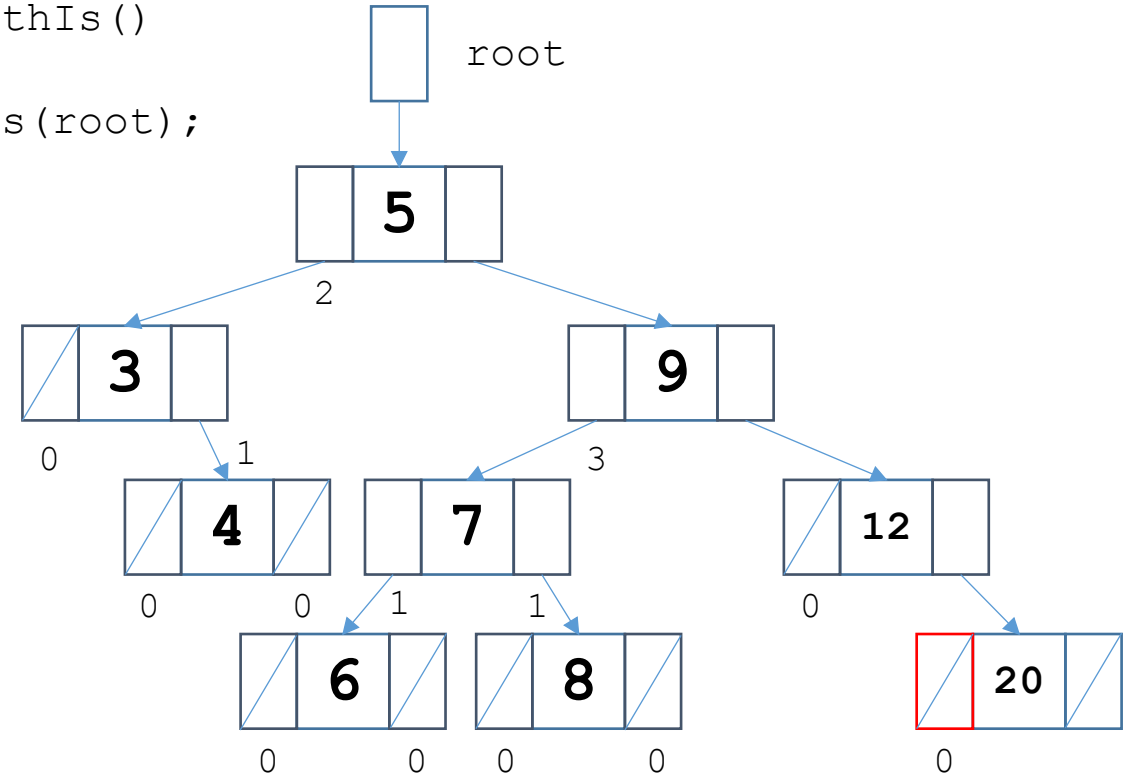
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

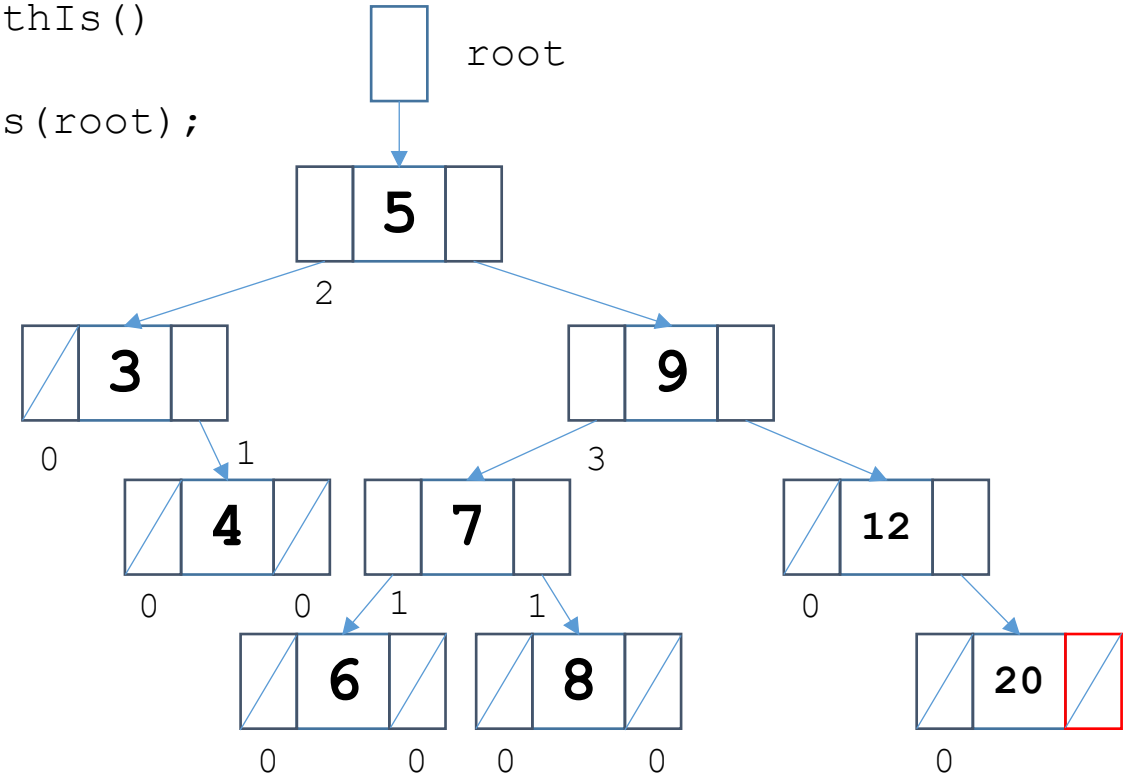


```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

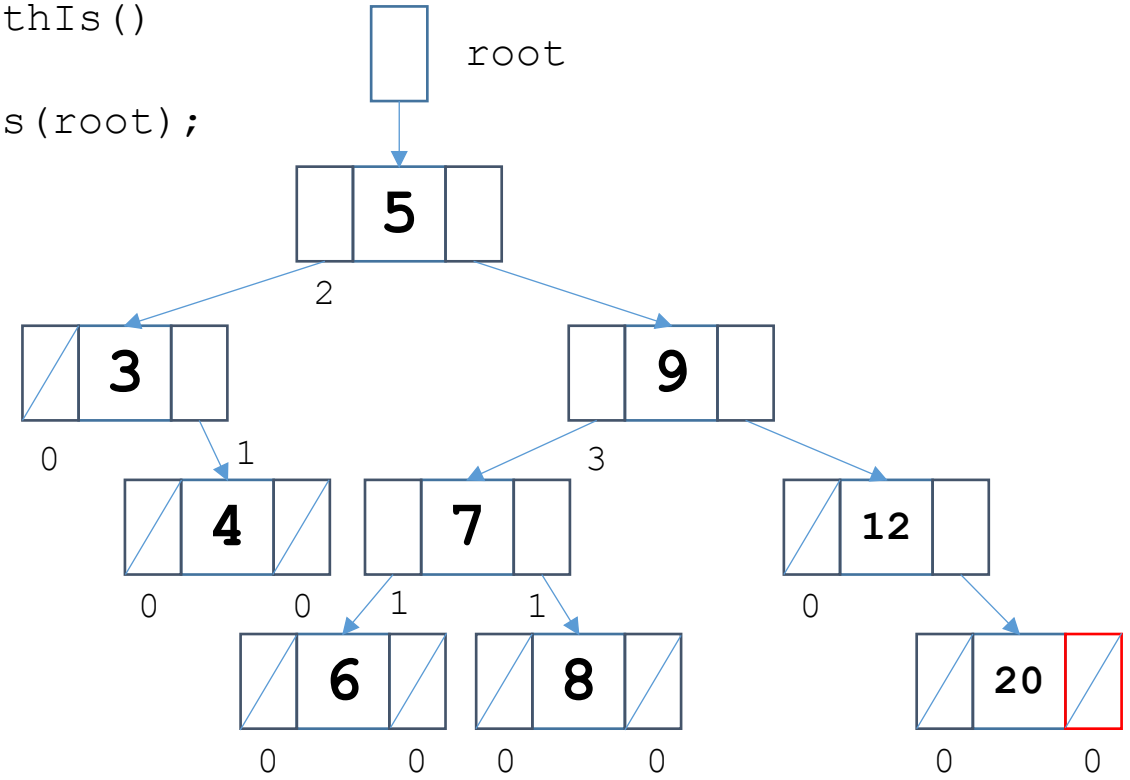


```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}

int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



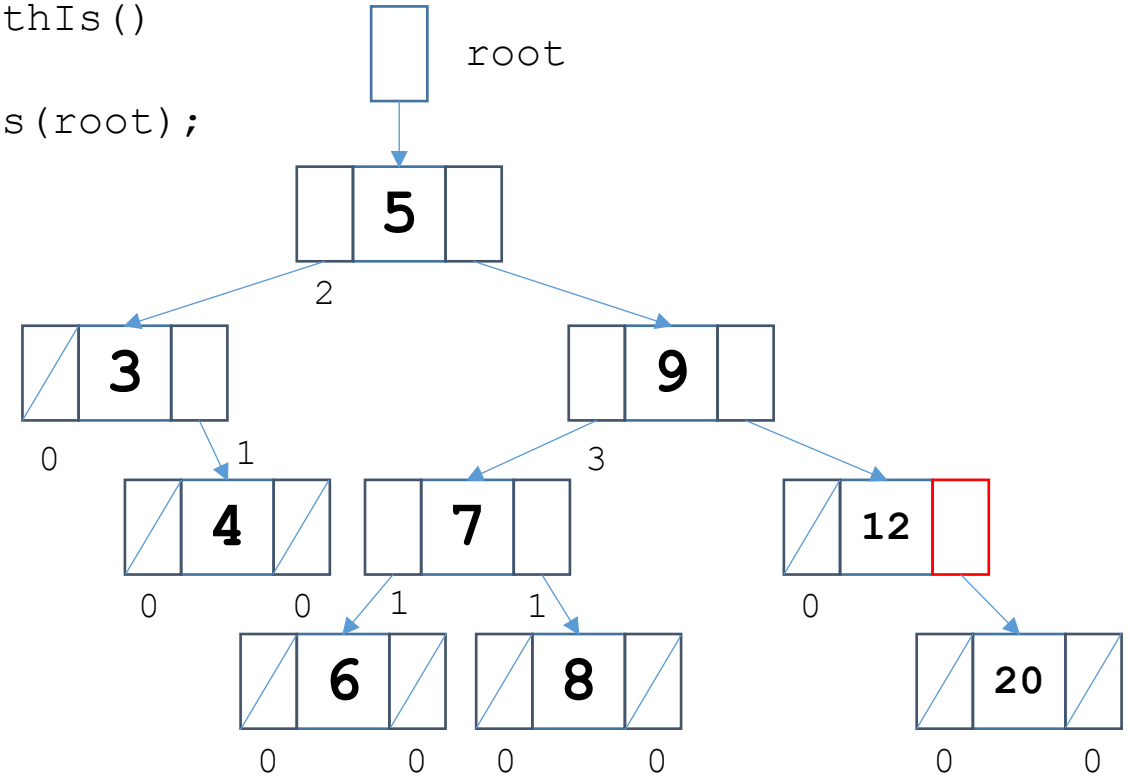
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



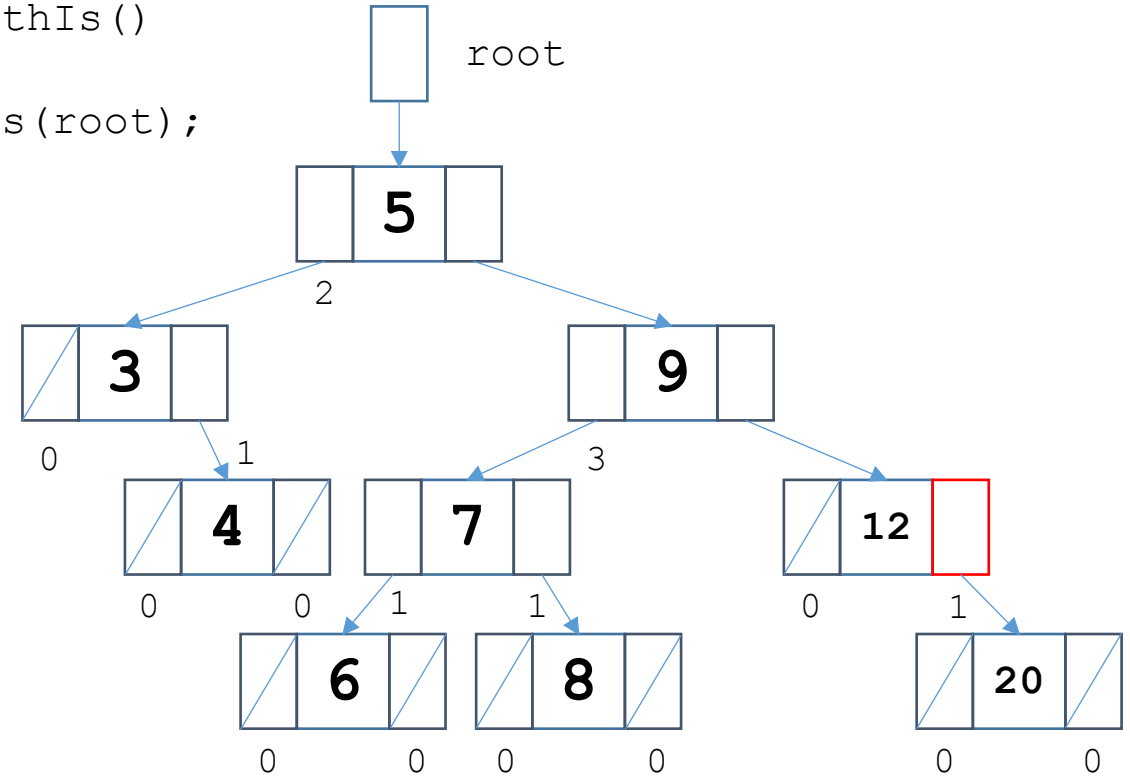
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



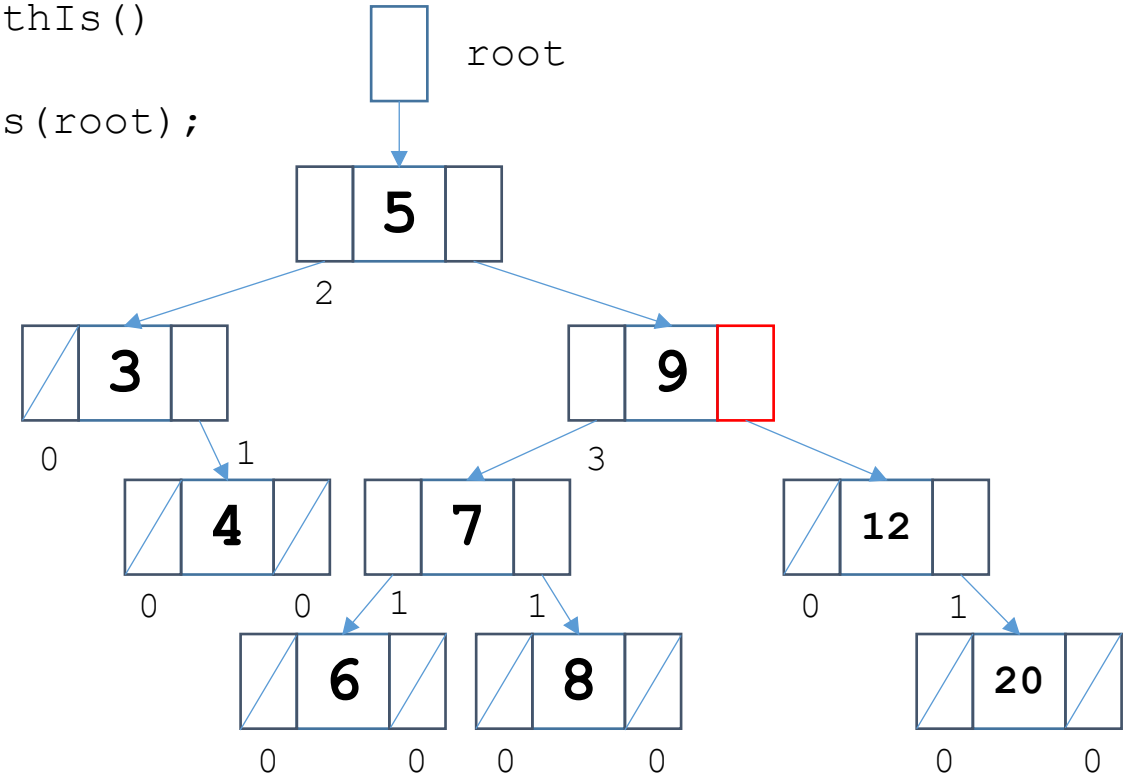
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



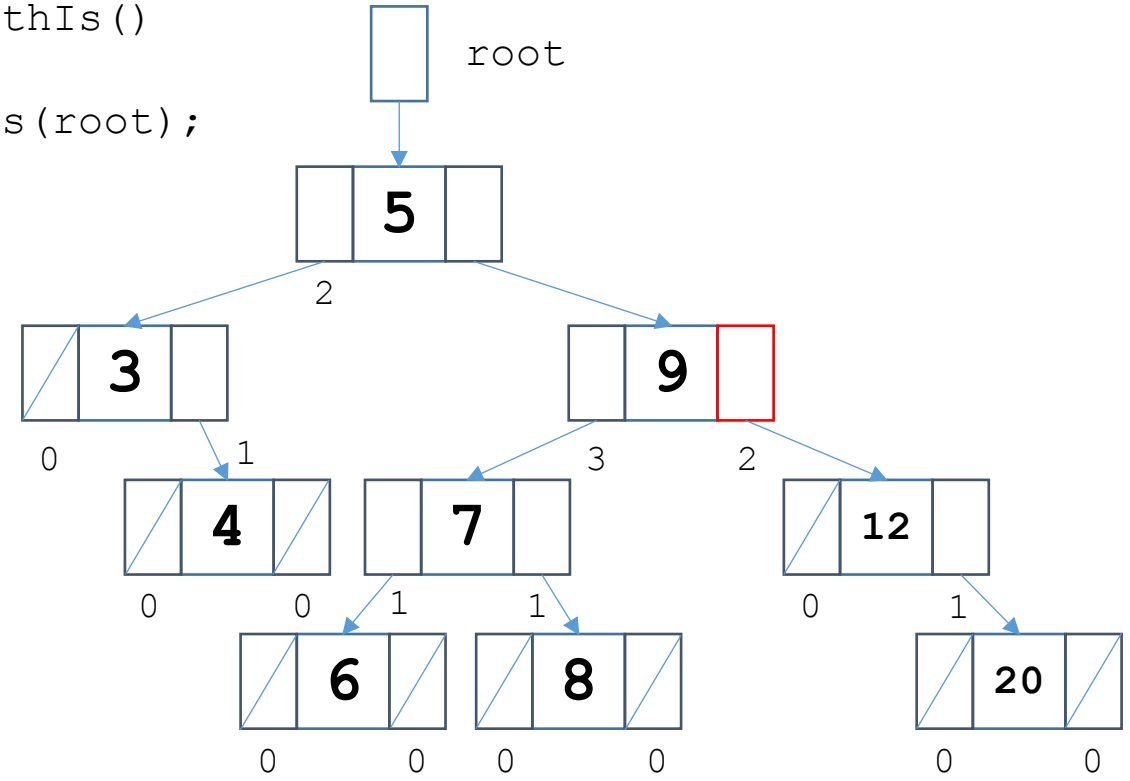
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



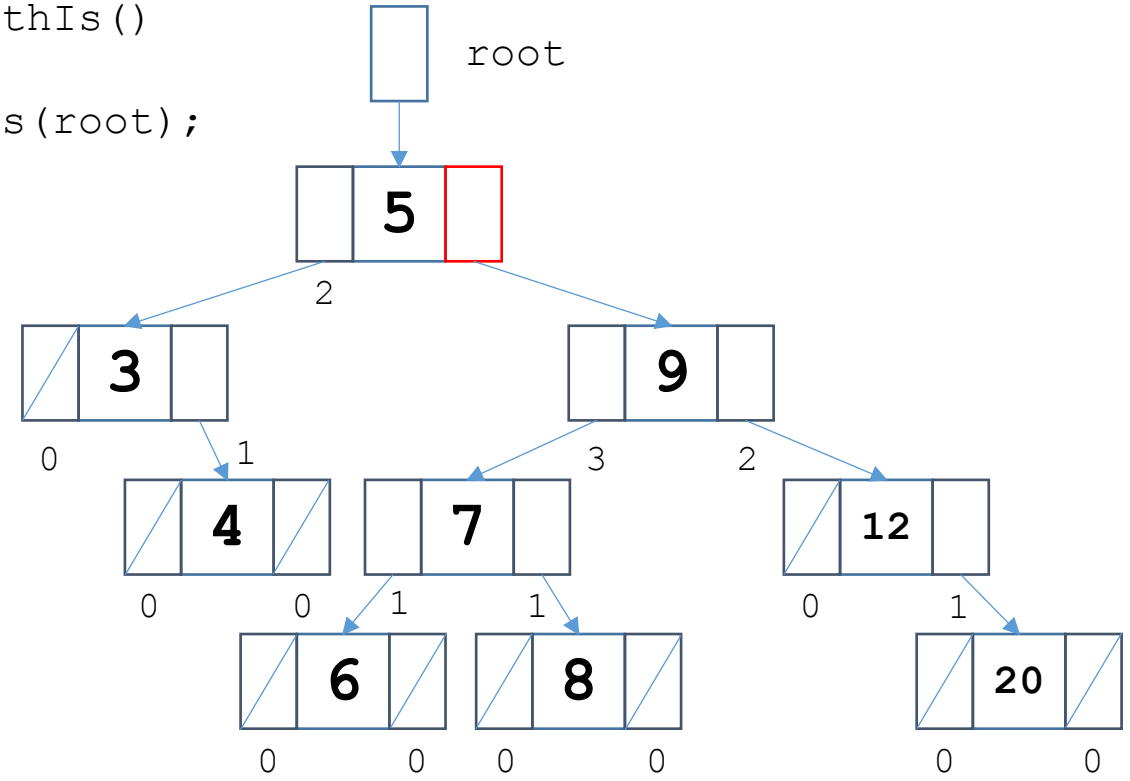
```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

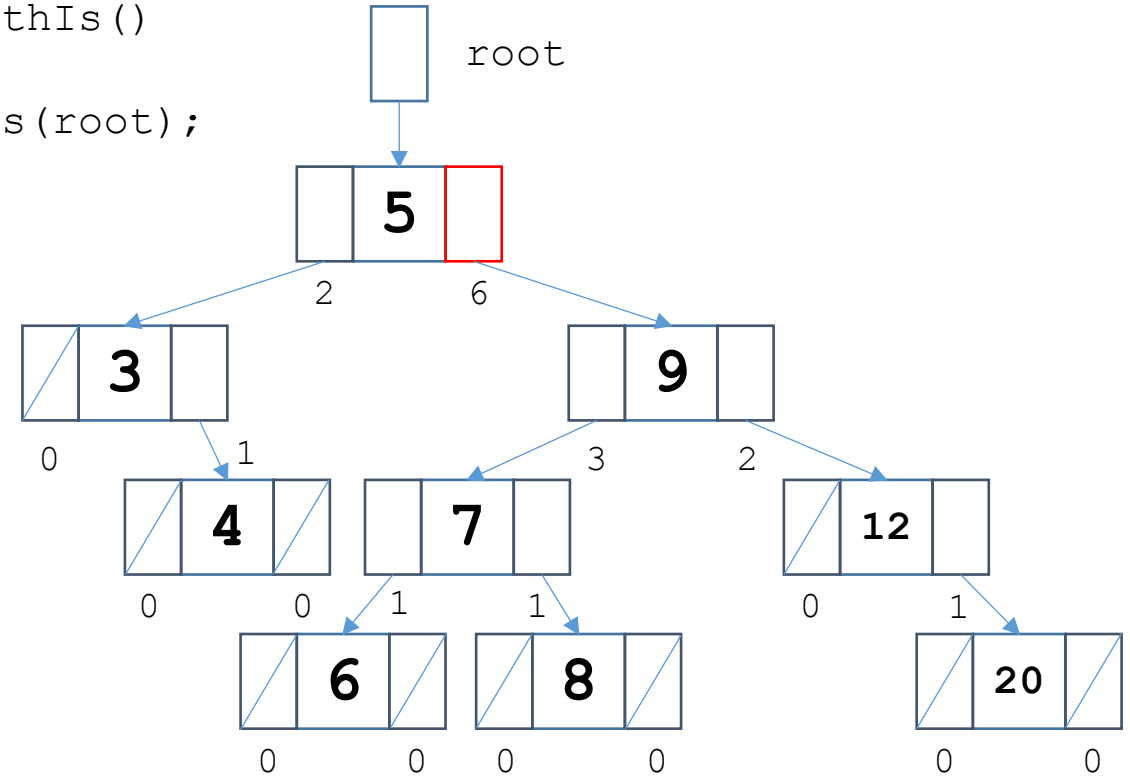


```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

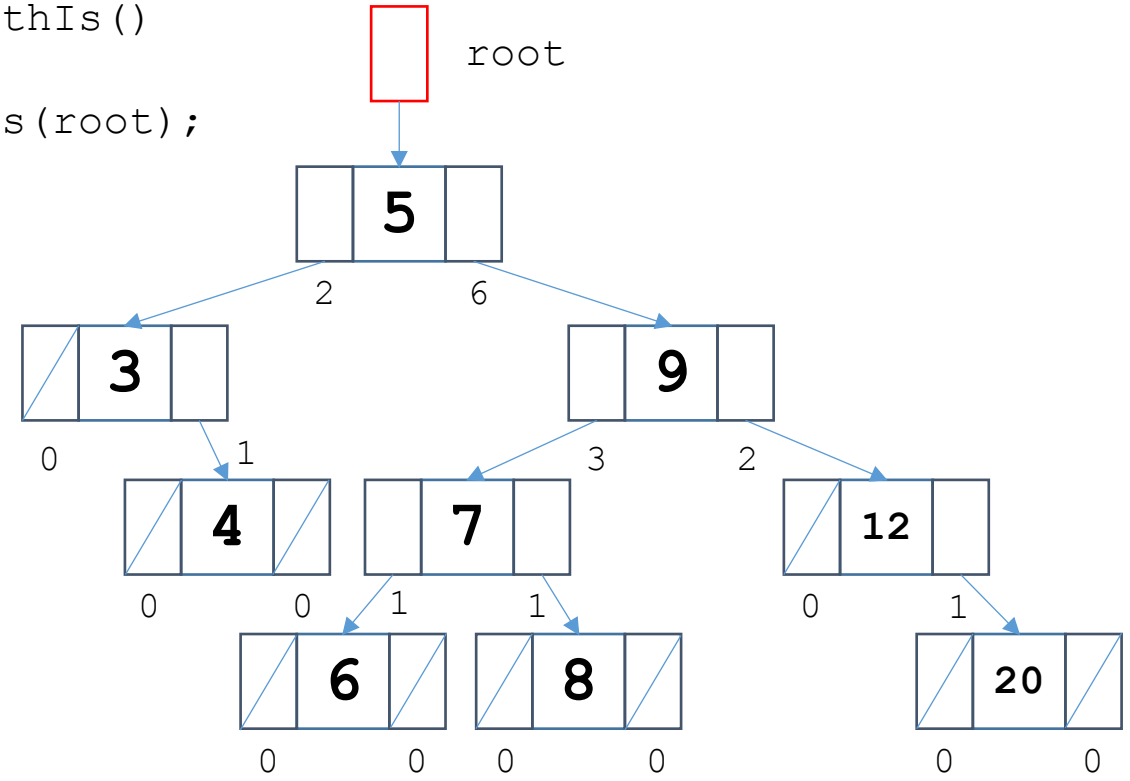


```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}

int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

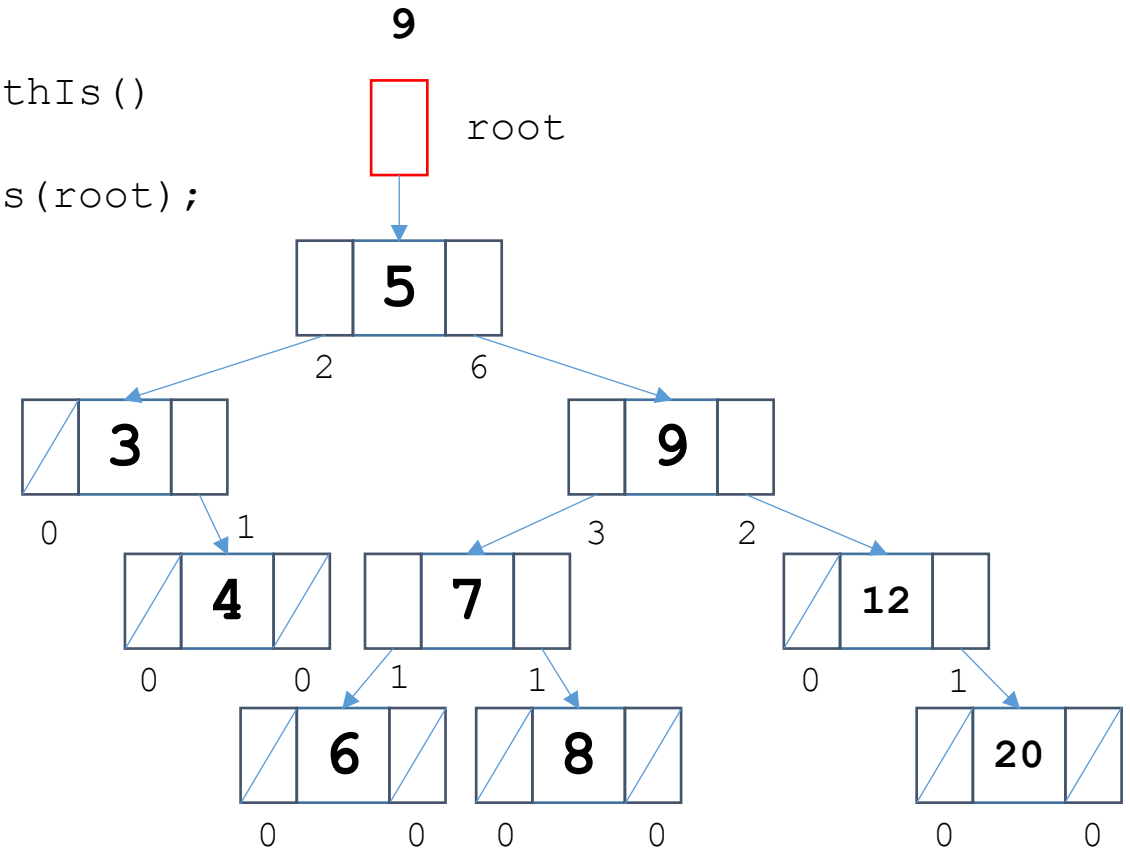


```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}
```

```
int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}

int TreeType::LengthIs()
{
  return CountNodes(root);
}
```



Function `LengthIs`

```
int CountNodes(TreeNode* tree)
{
  if (tree == NULL)
    return 0;
  else
    return CountNodes(tree->left) + CountNodes(tree->right) + 1;
}

int TreeType::LengthIs()
{
  return CountNodes(root);
}
```

O(N)

Function `RetrieveItem`

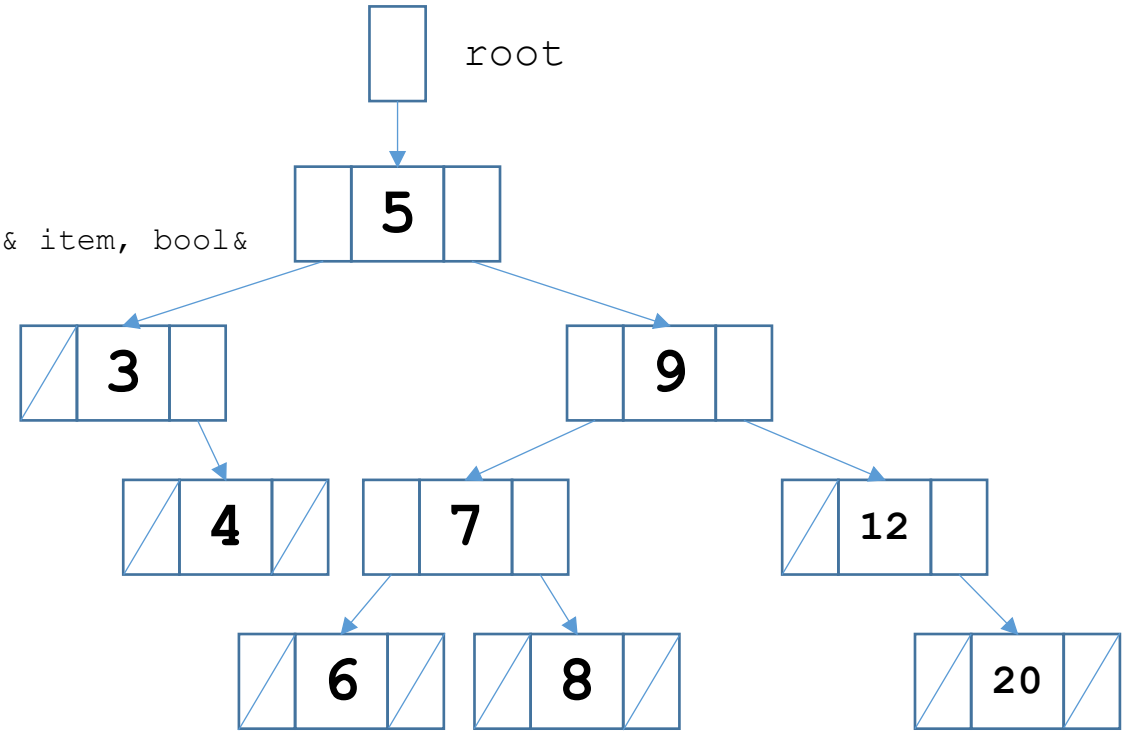
- Find the target node recursively
  - If tree is empty (base case 1)
    - Item is not found in the tree
  - Item == current_node_info (base case 2)
    - Item is found
  - Item < current_node_info (general case 1)
    - Search the left subtree
  - Item > current_node_info (general case 2)
    - Search the right subtree

Function `RetrieveItem`

```
void Retrieve(TreeNode* tree, ItemType& item, bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool& found)
{
    Retrieve(root, item, found);
}
```

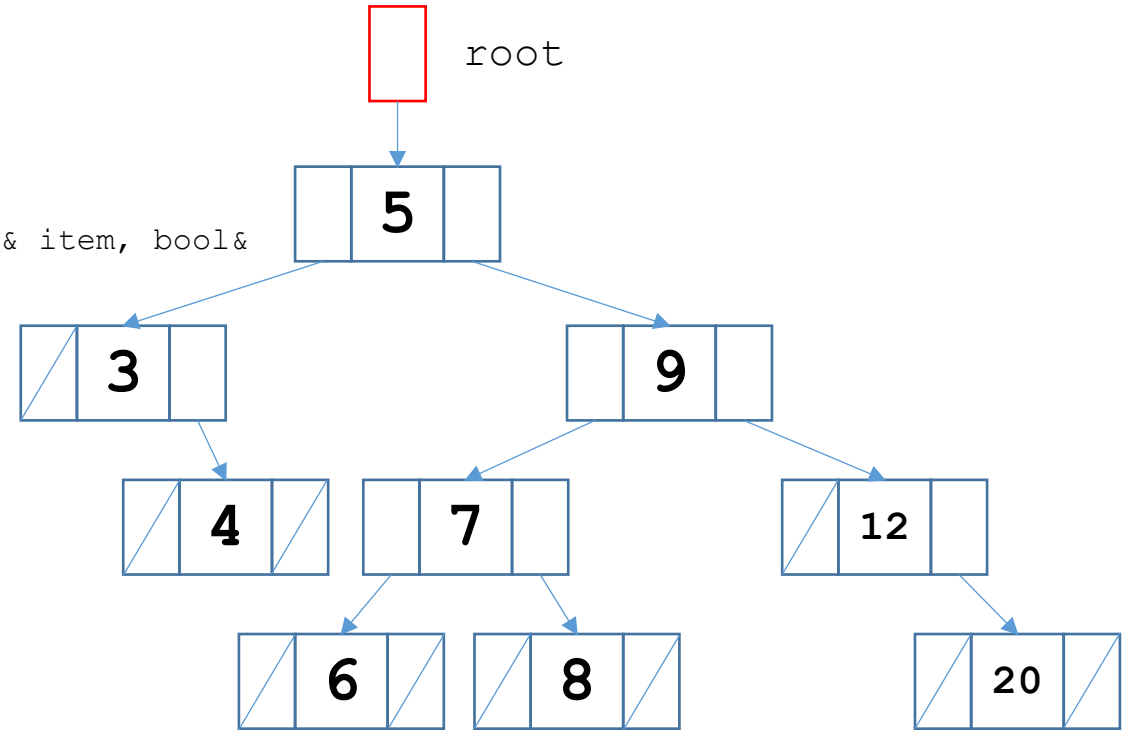
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



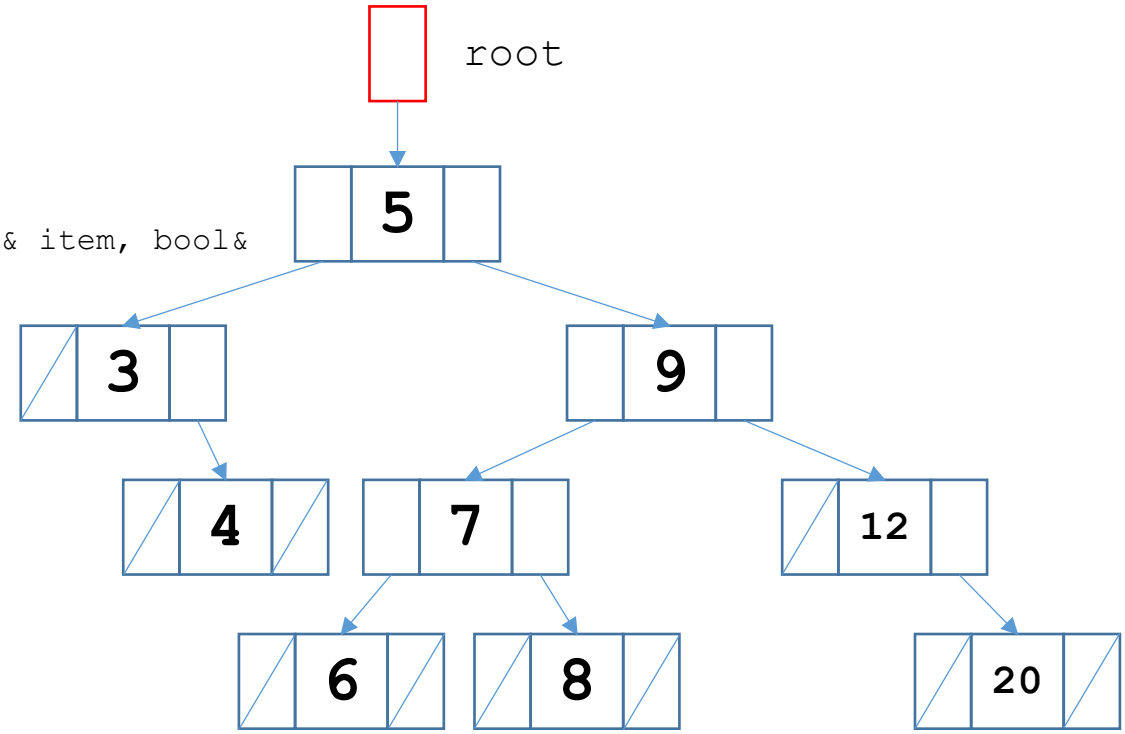
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



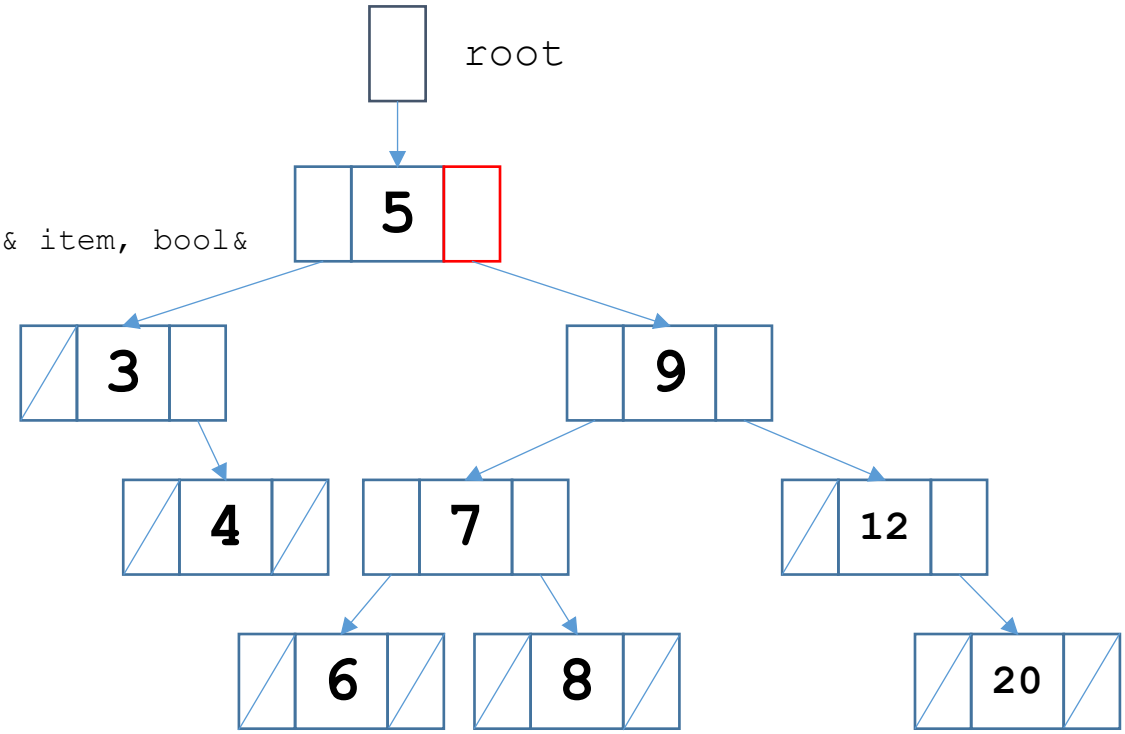
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



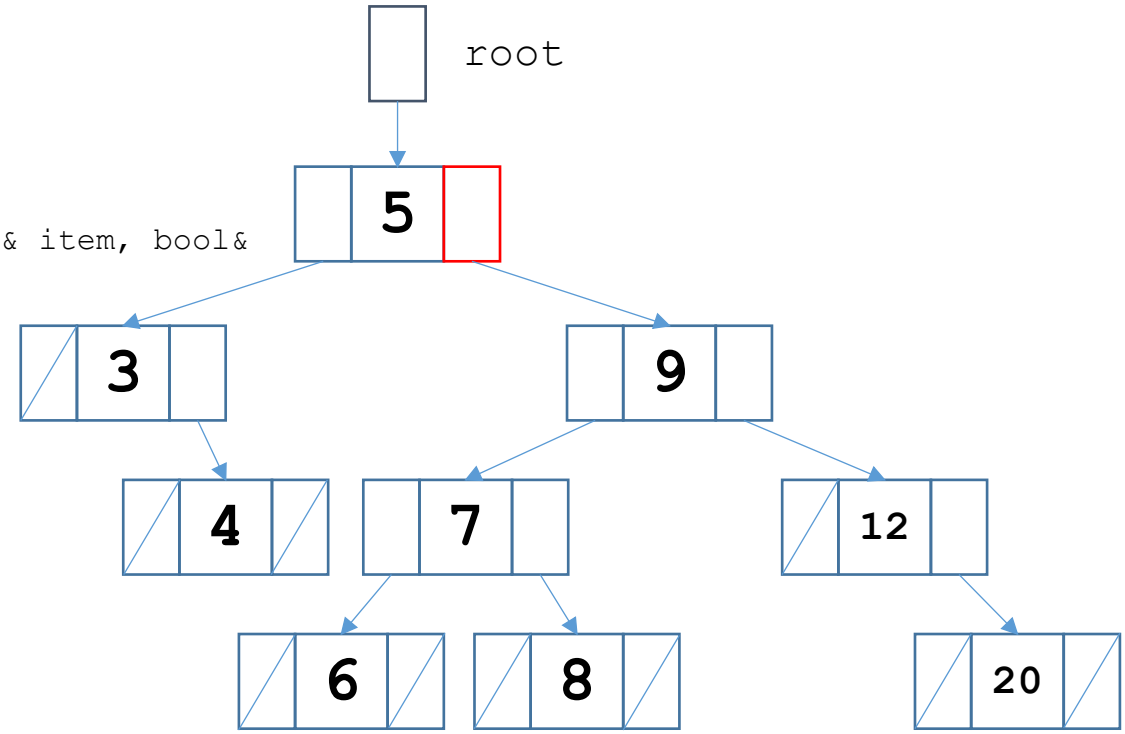
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



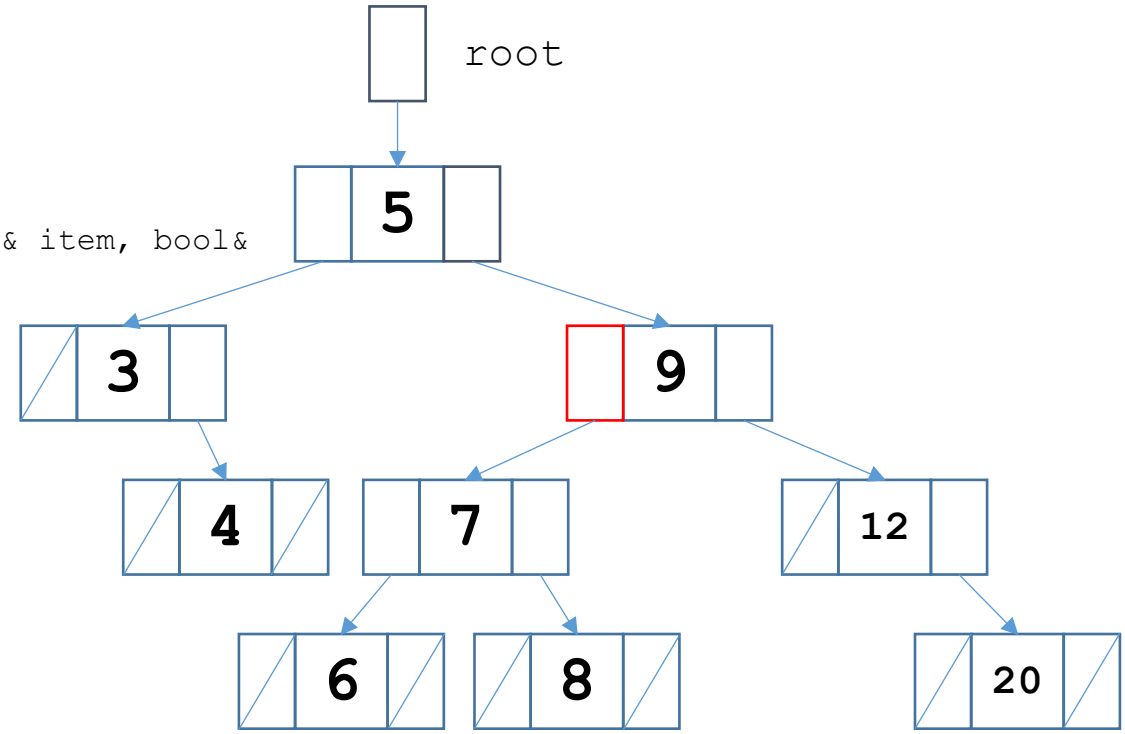
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



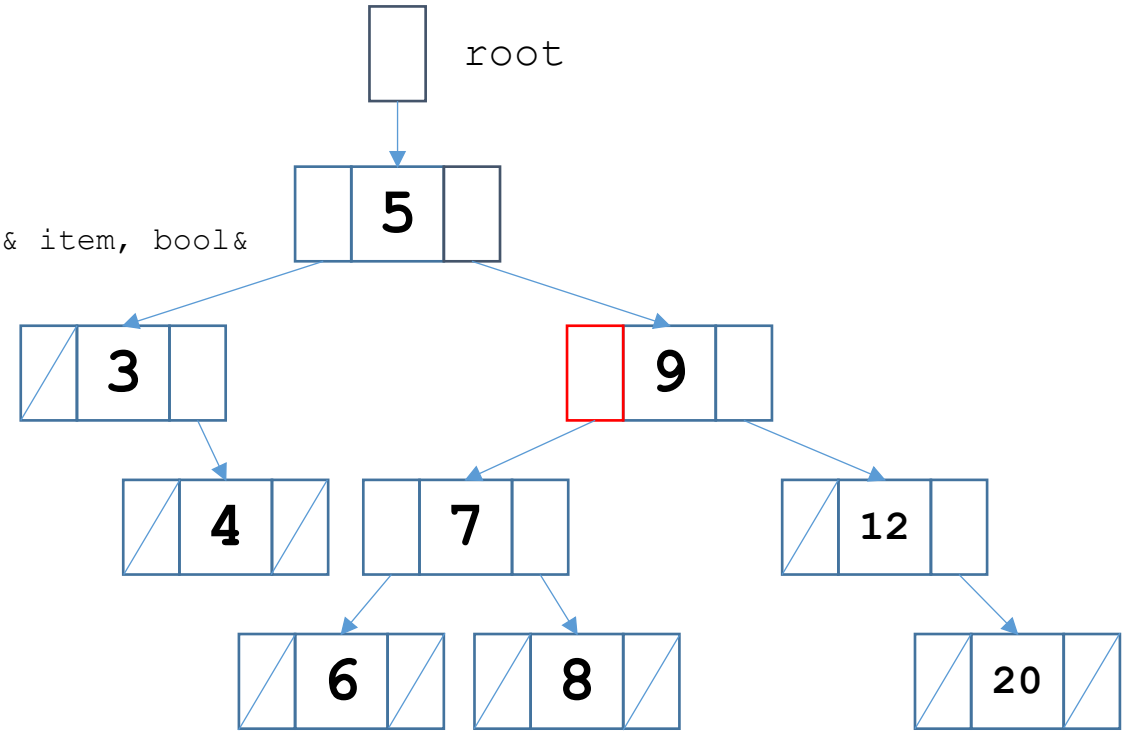
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



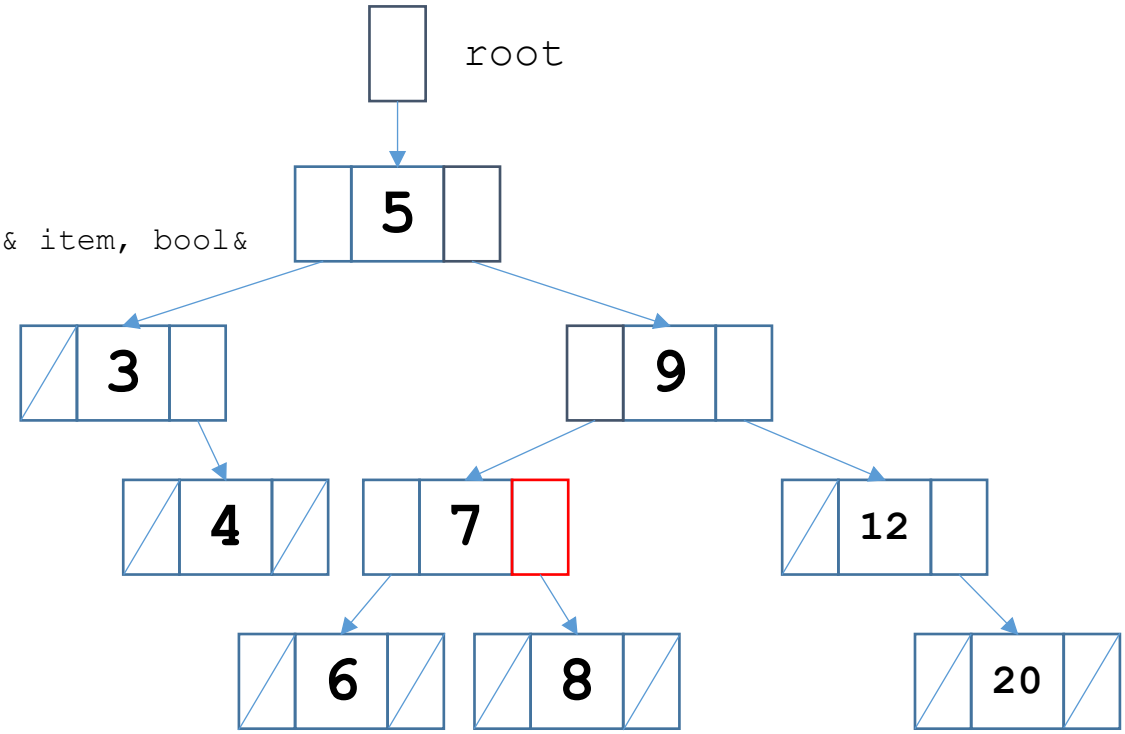
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8



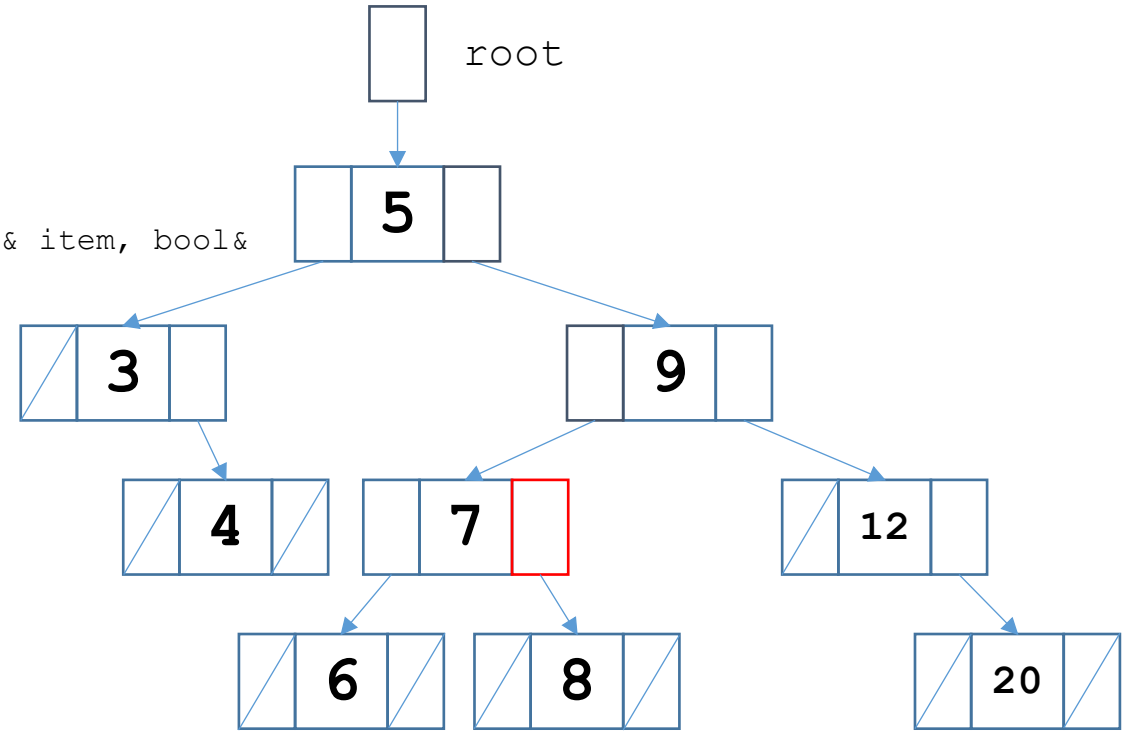
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8

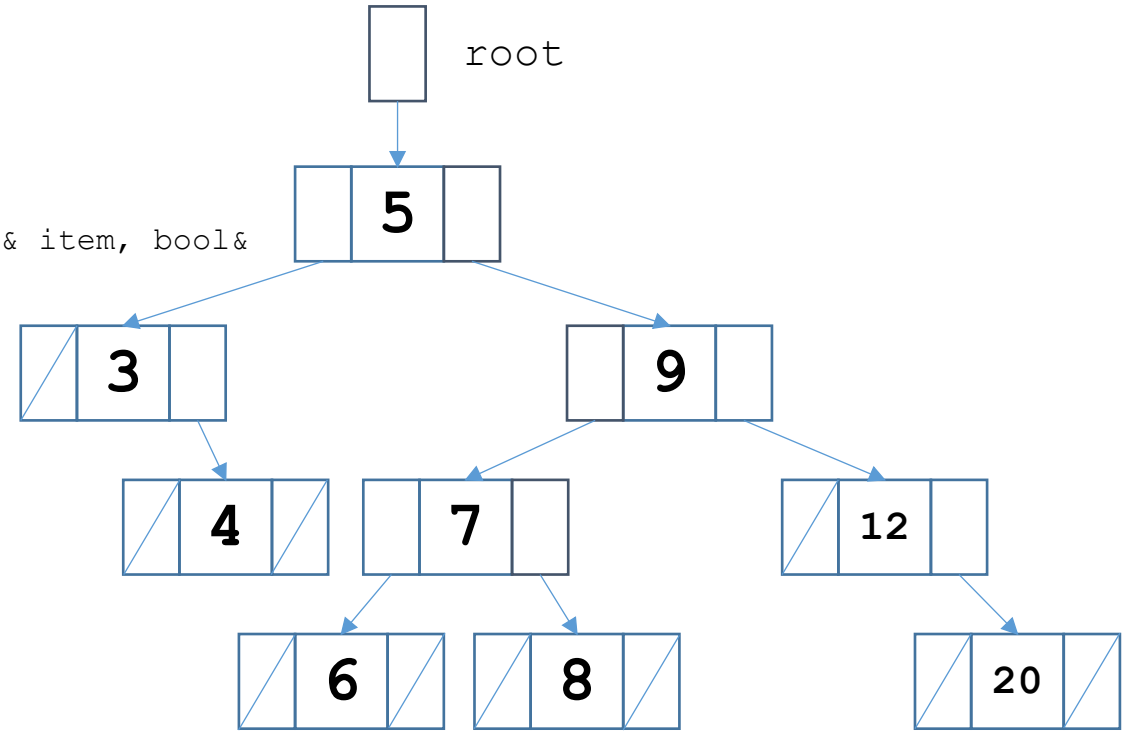


```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 8

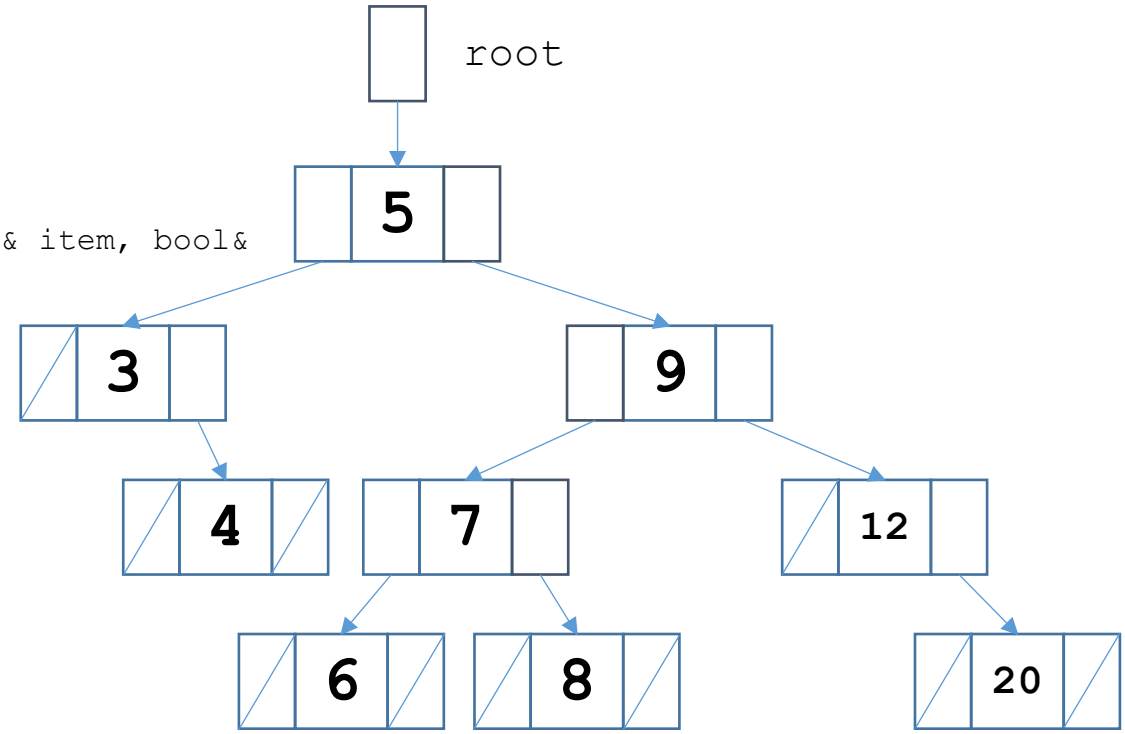


```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```



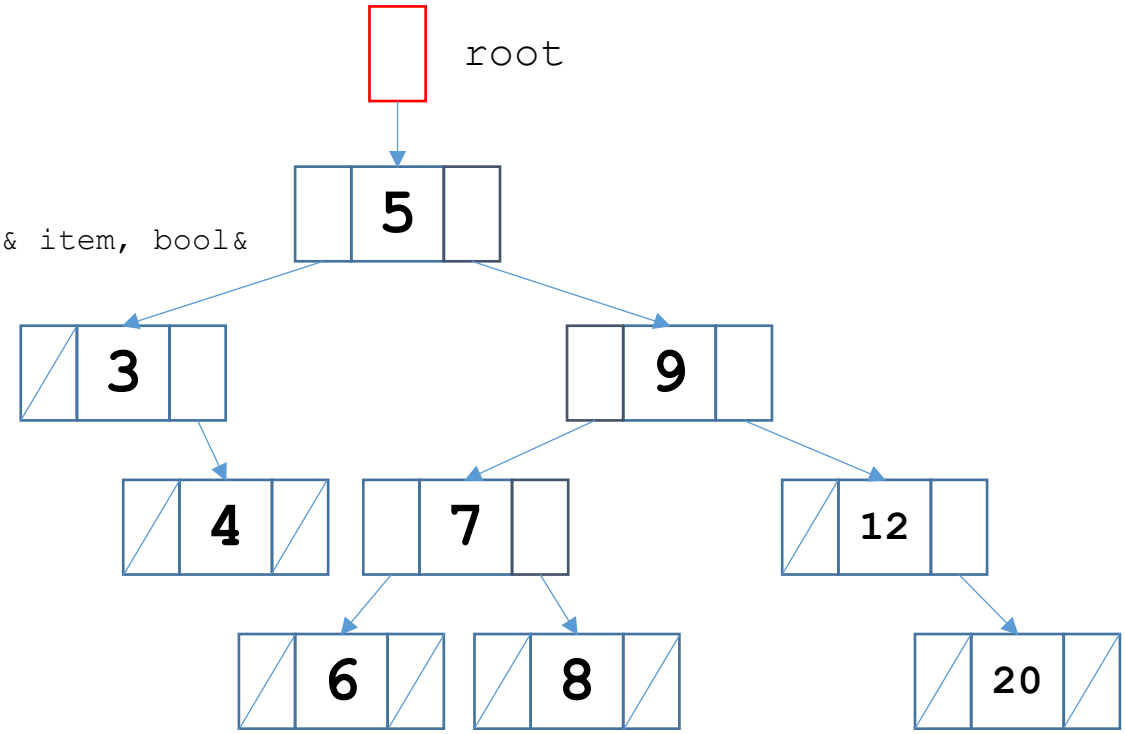
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



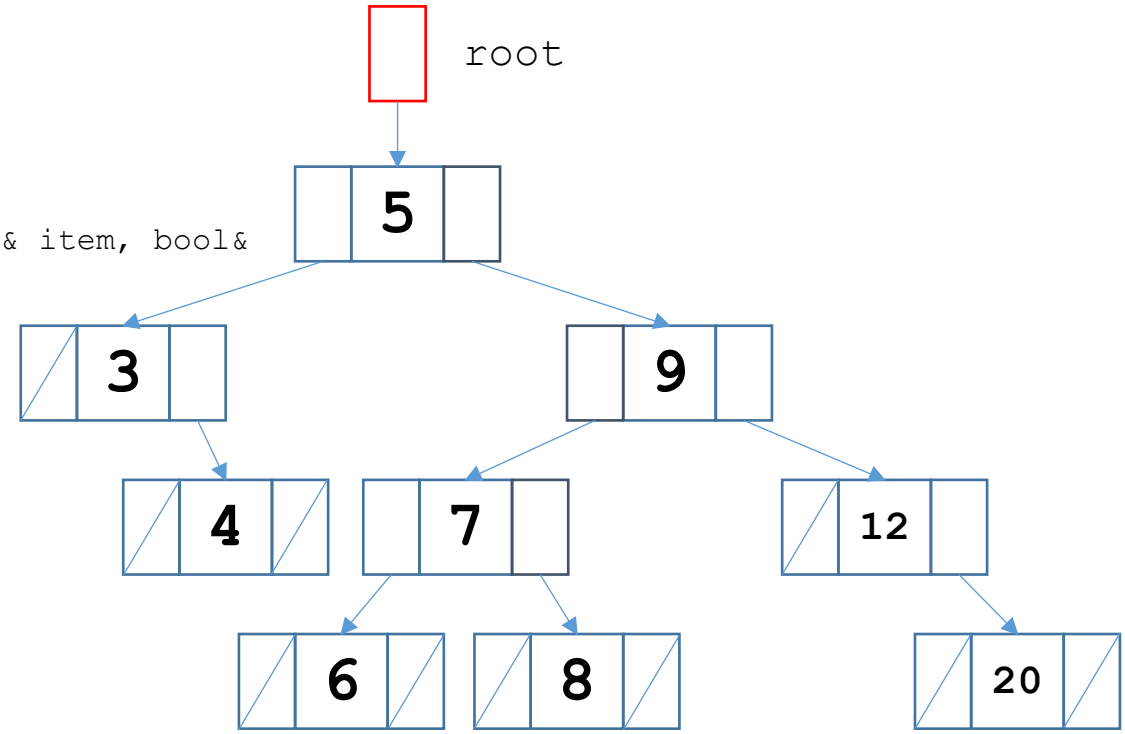
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



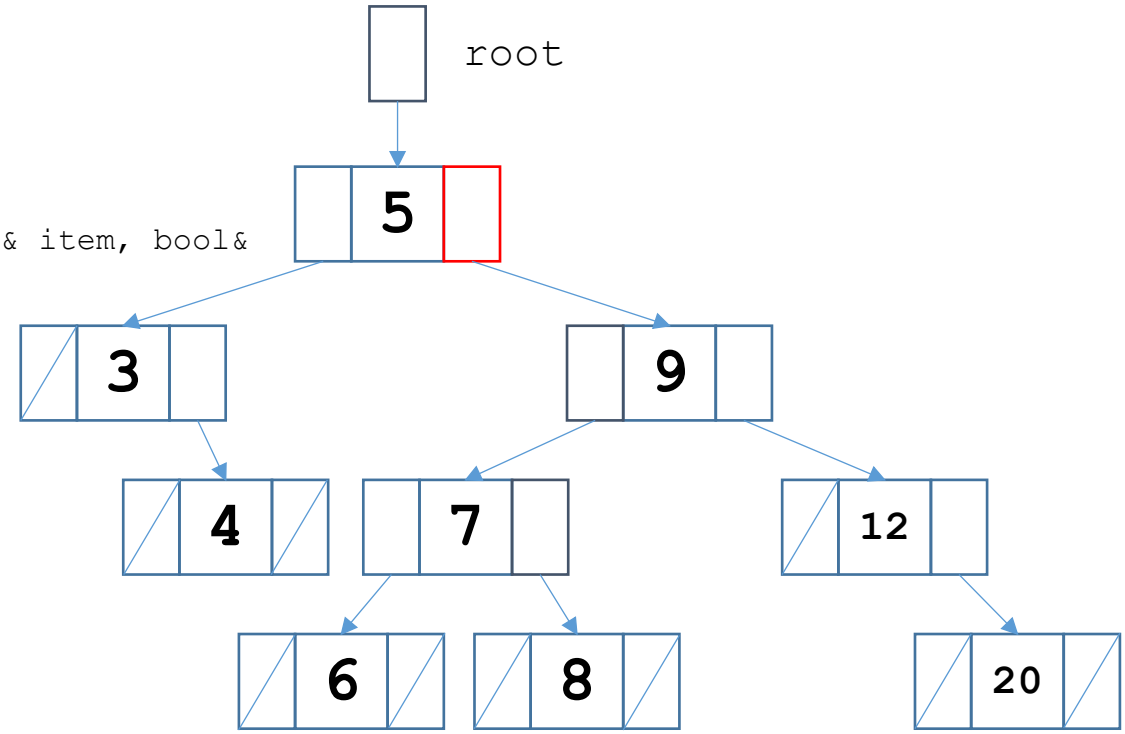
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



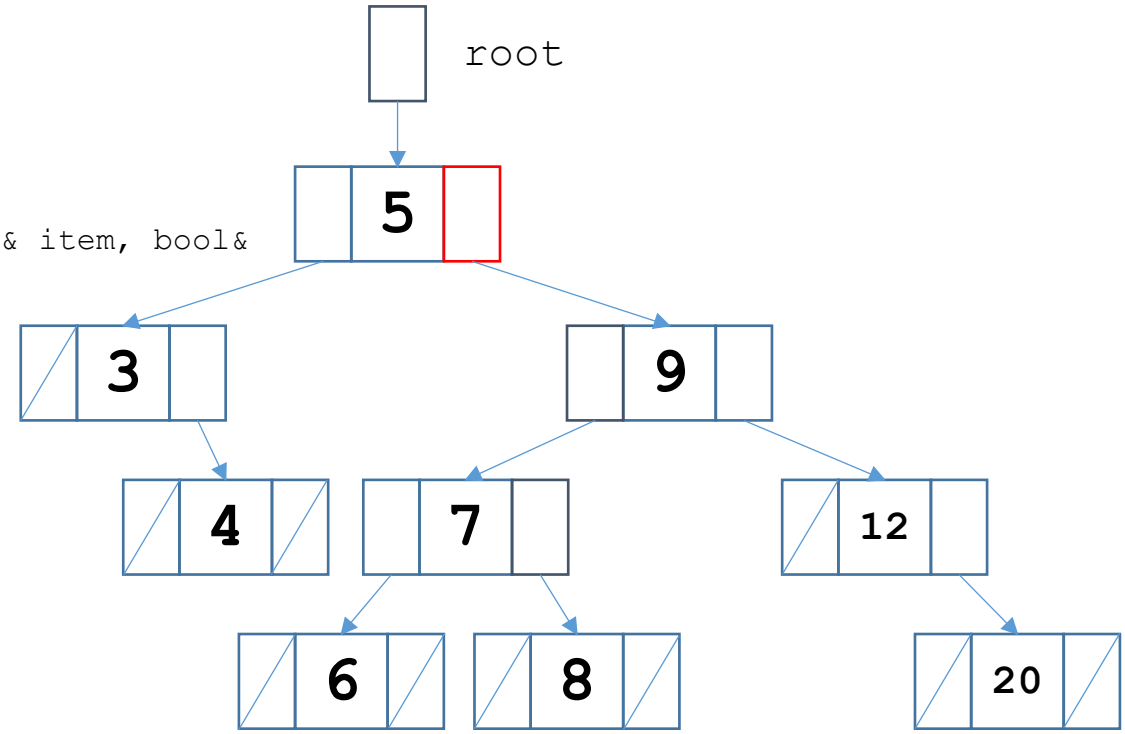
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



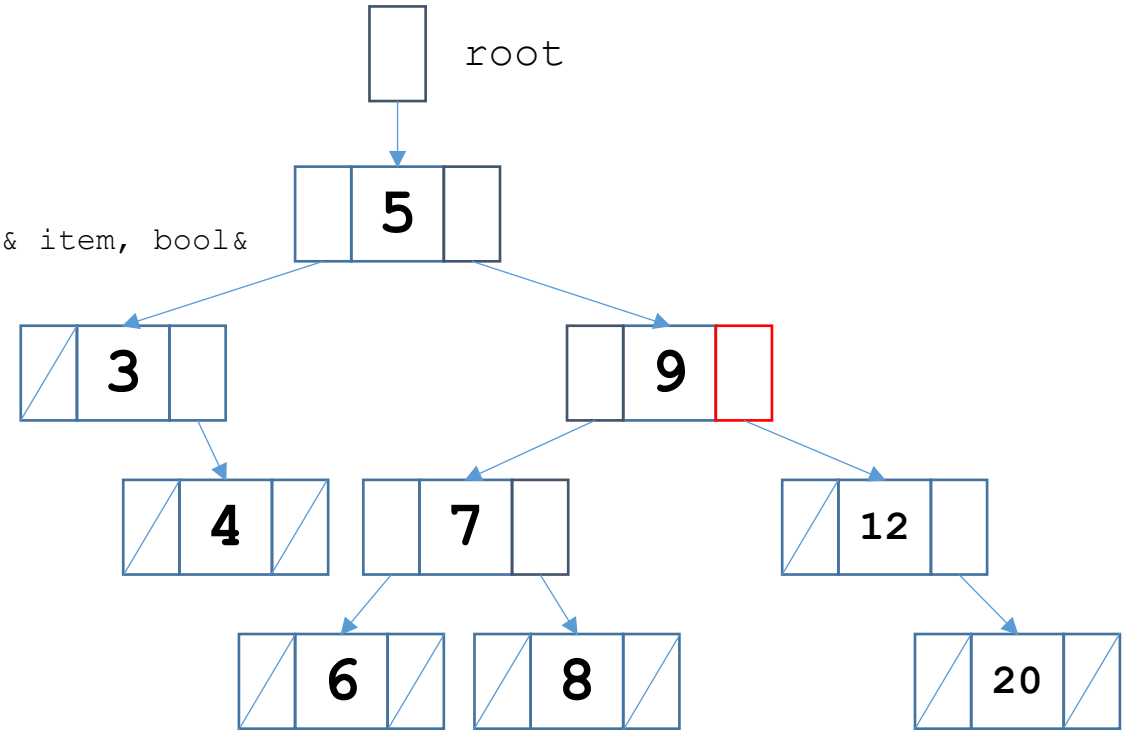
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



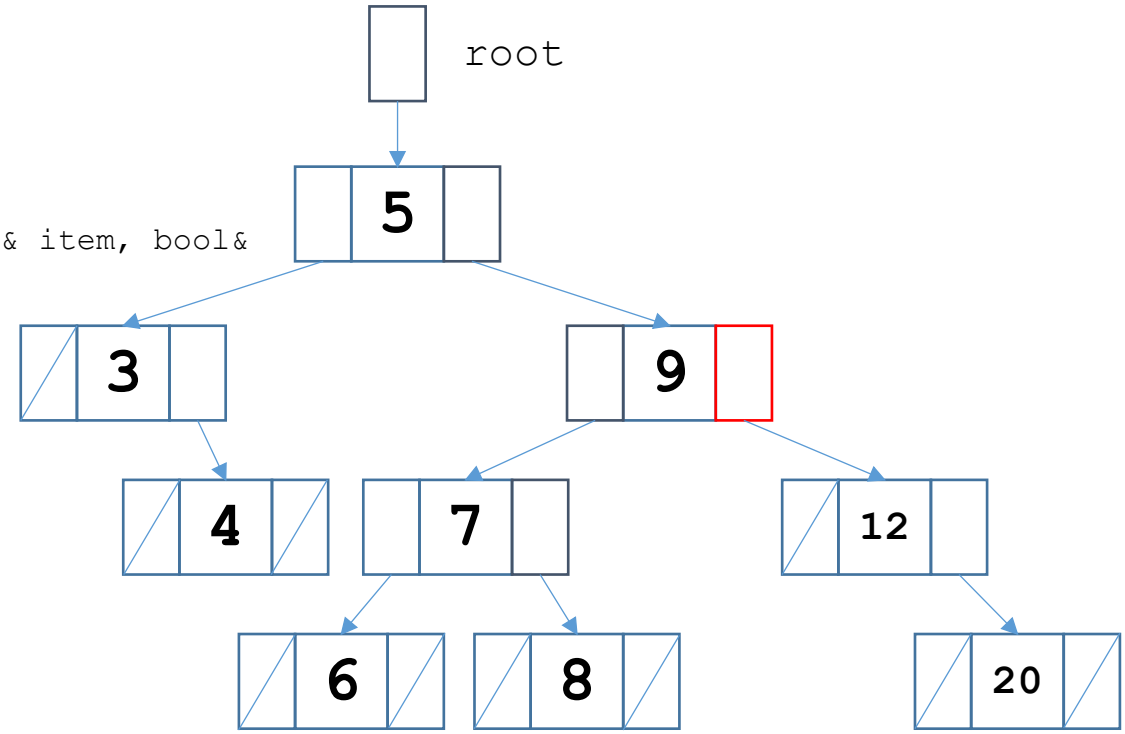
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



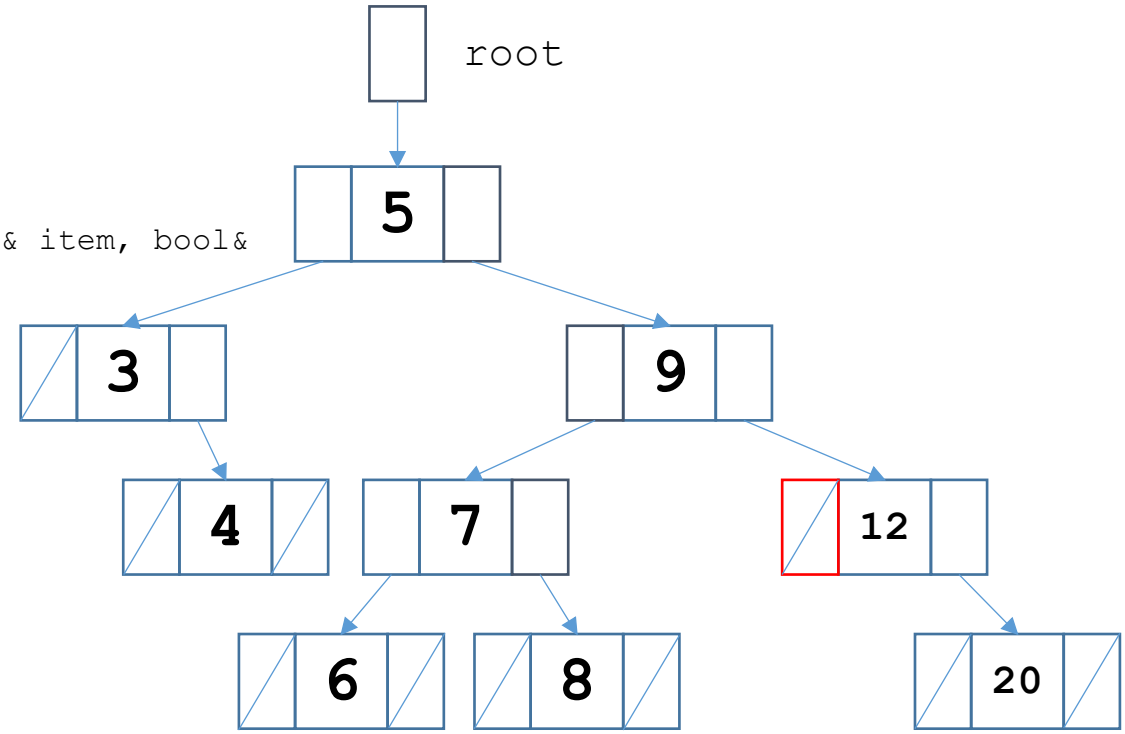
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



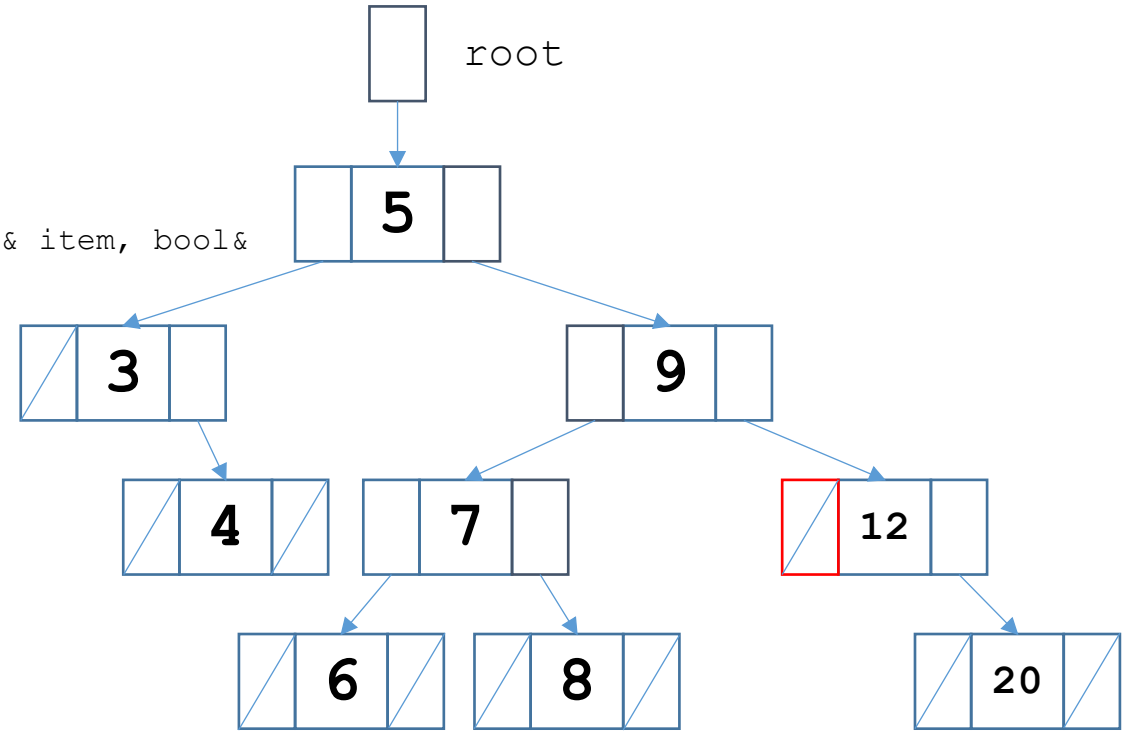
```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



```
void Retrieve(TreeNode* tree, ItemType& item,
bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool&
found)
{
    Retrieve(root, item, found);
}
```

Find 10



Function `RetrieveItem`

```
void Retrieve(TreeNode* tree, ItemType& item, bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}

void TreeType::RetrieveItem(ItemType& item, bool& found)
{
    Retrieve(root, item, found);
}
```

Worst case: **O(N)**

Best case: **O(logN)**

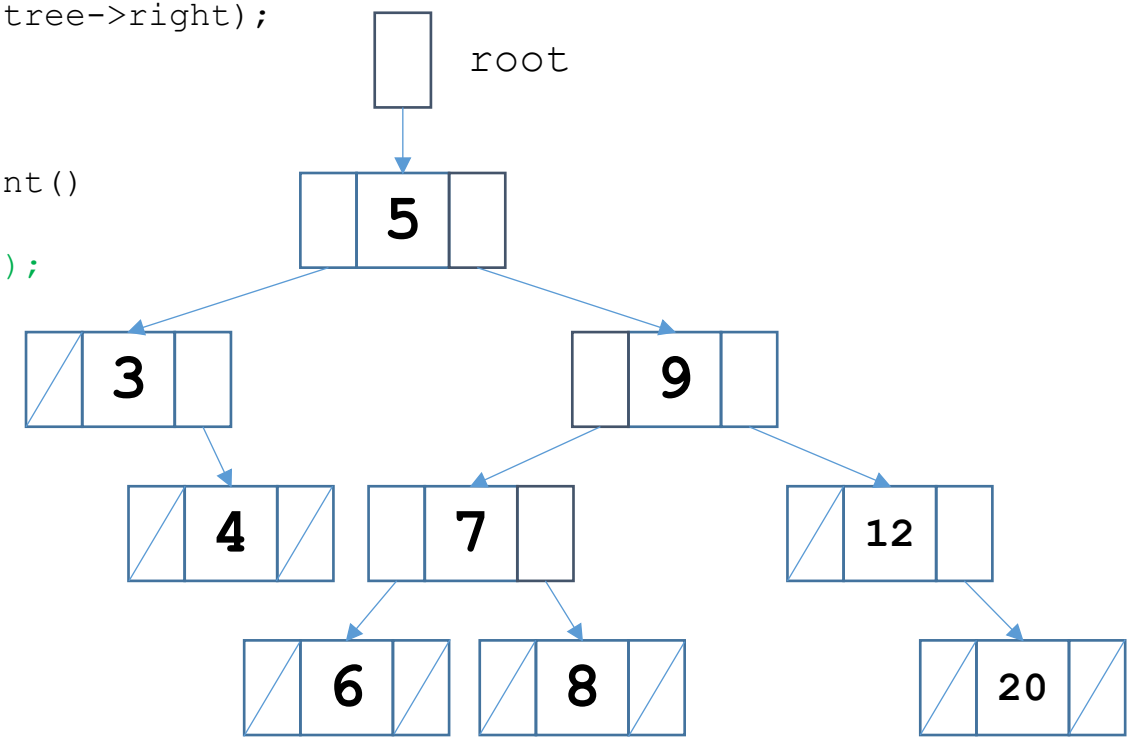
Function `Print`

- Print the items in the tree (**in sorted order**) recursively
  - If tree is empty (base case)
    - Nothing to print
  - If tree is non-empty (general case)
    - Print the items in the left subtree
    - Print the item at the current node
    - Print the items in the right subtree

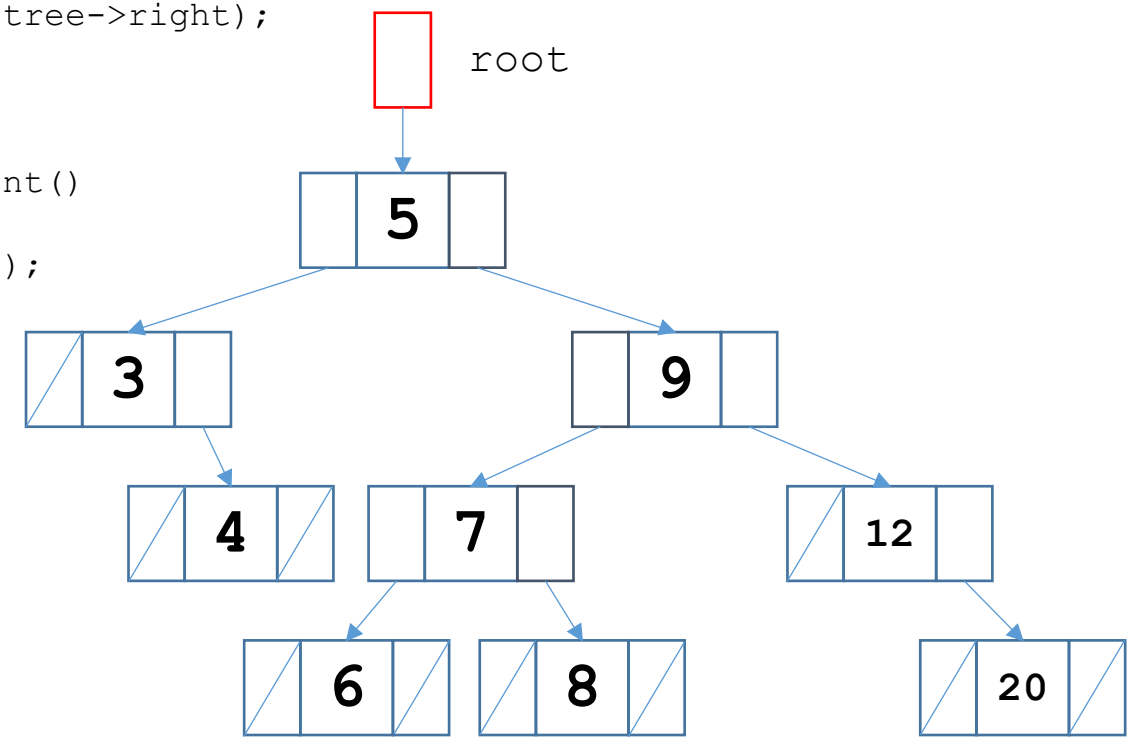
Function `Print`

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info << endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```

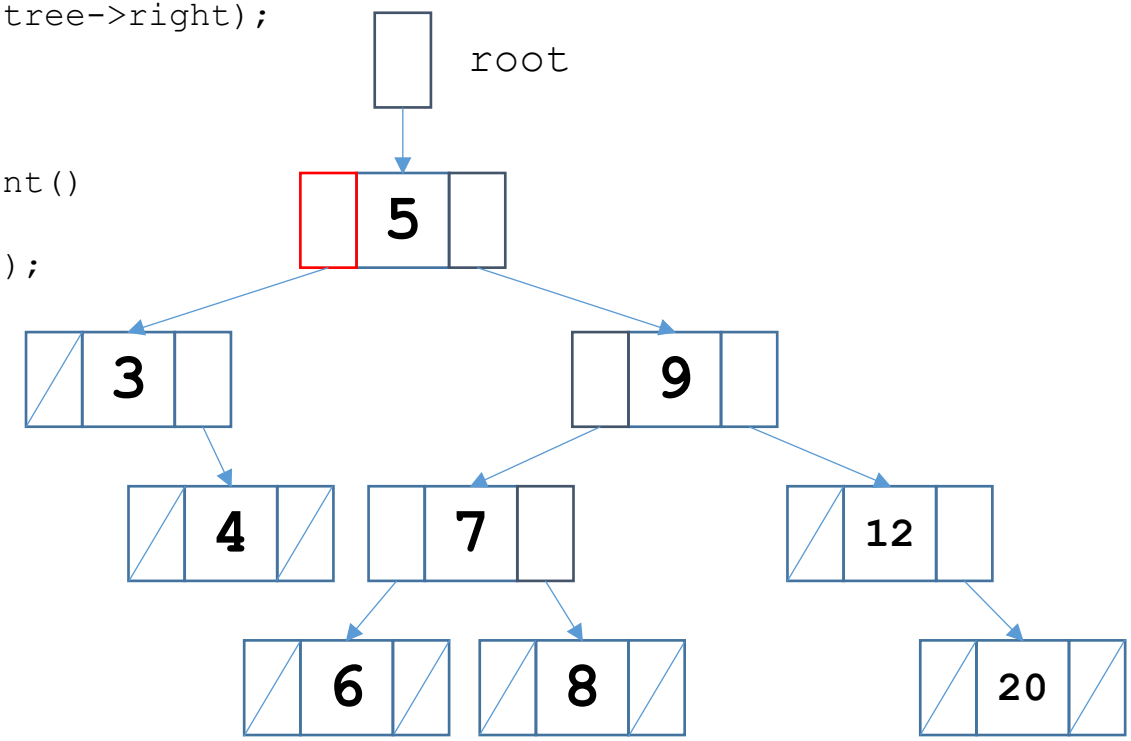
```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



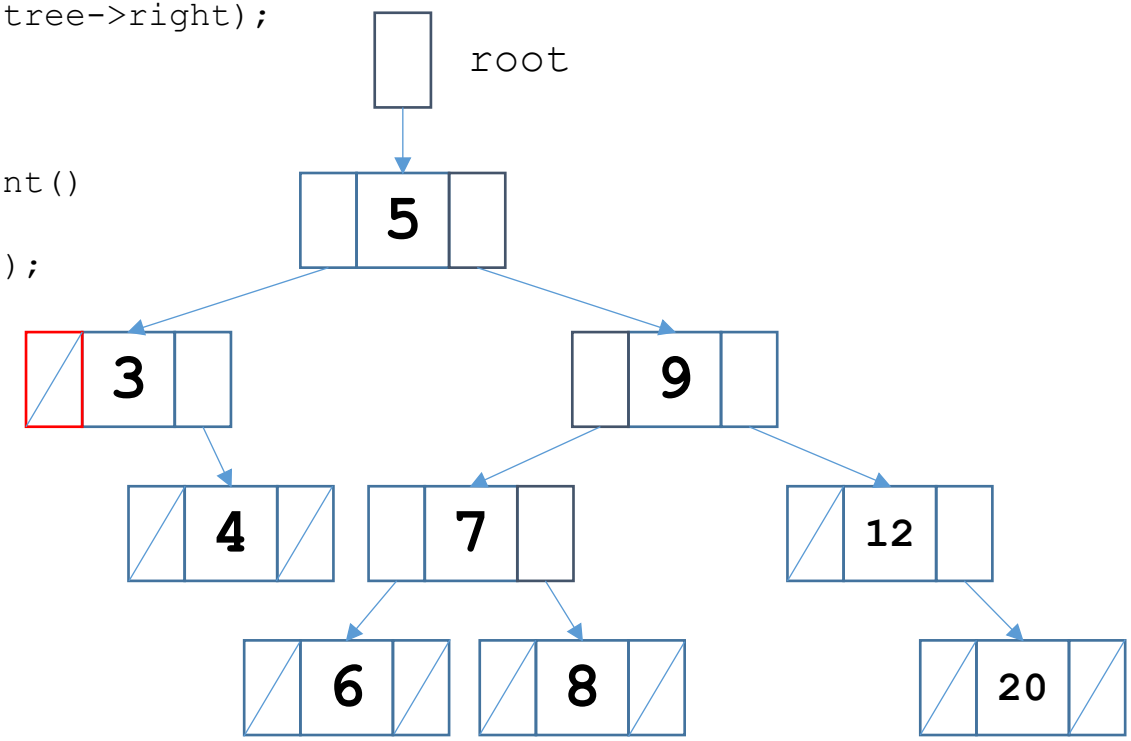
```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



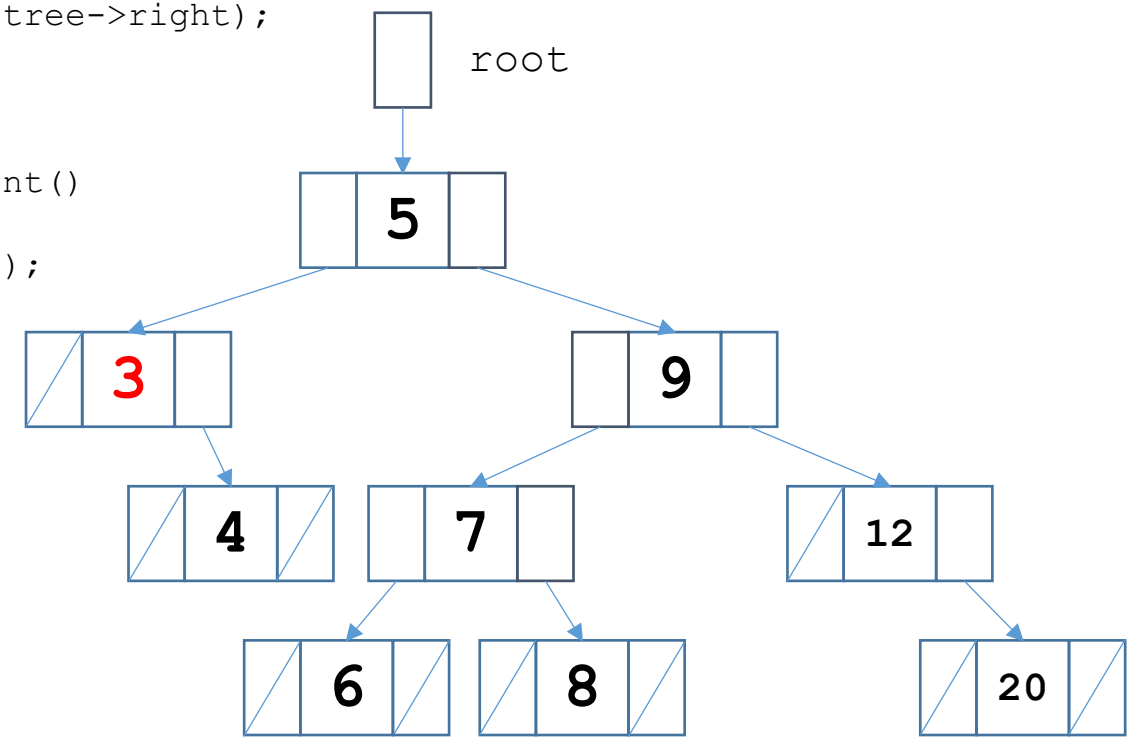
```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```

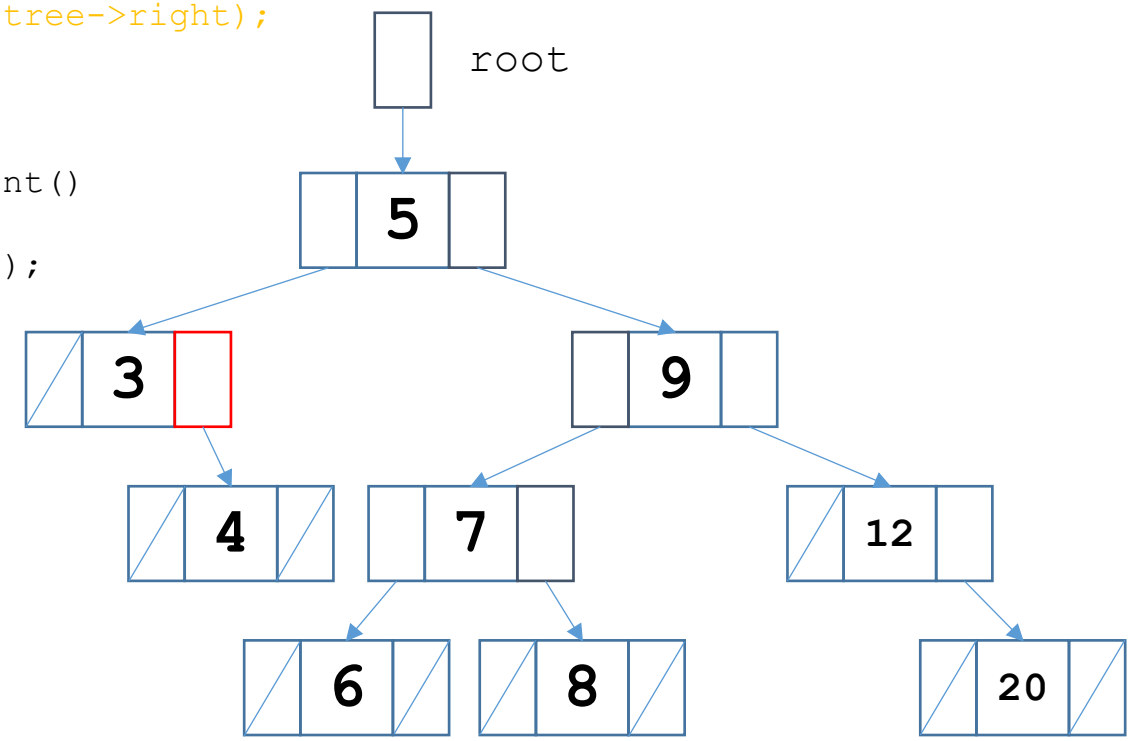


```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



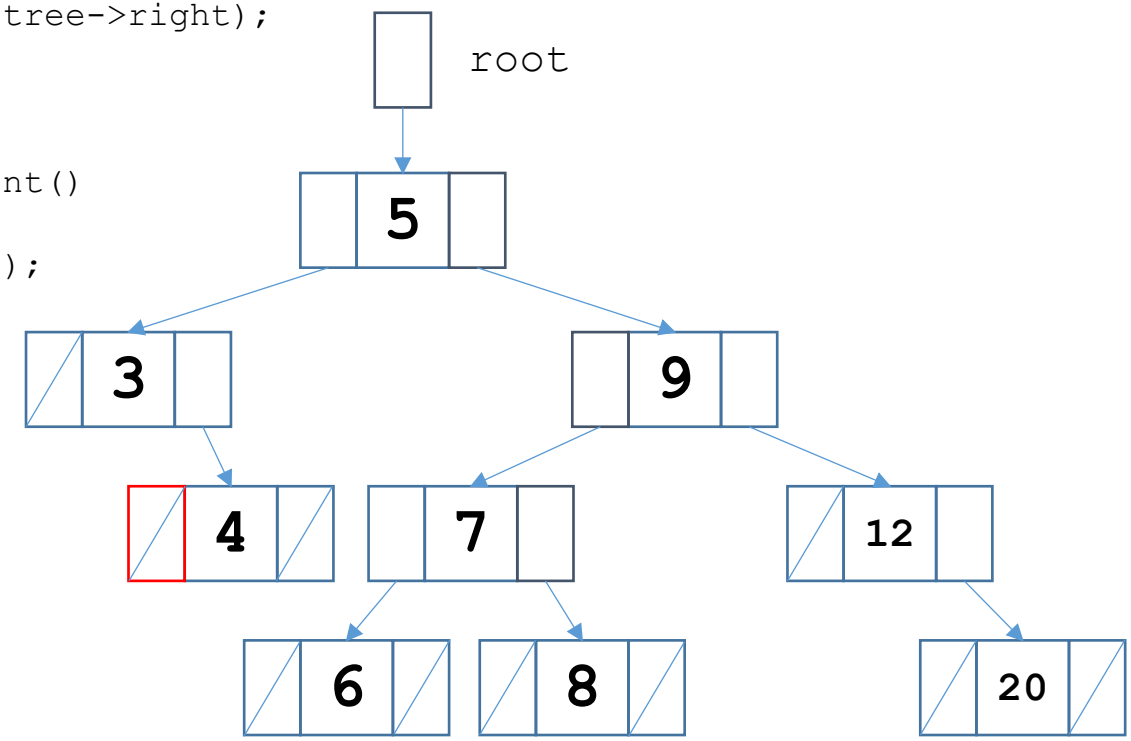
3

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



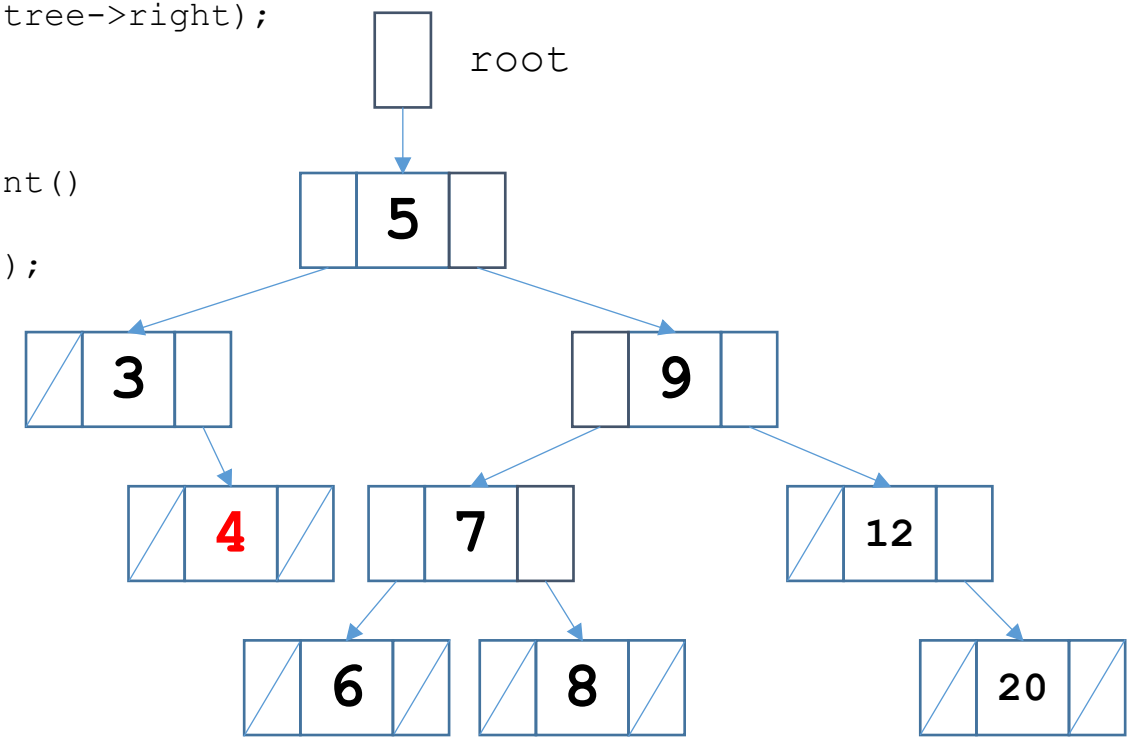
3

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



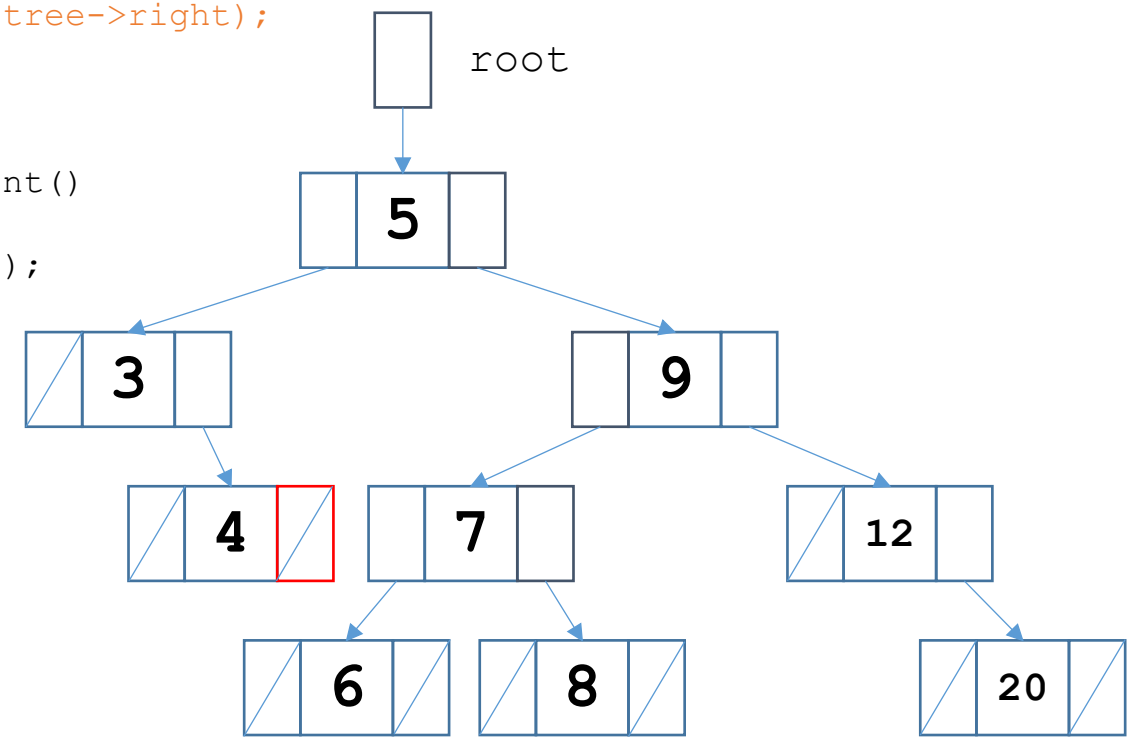
3

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



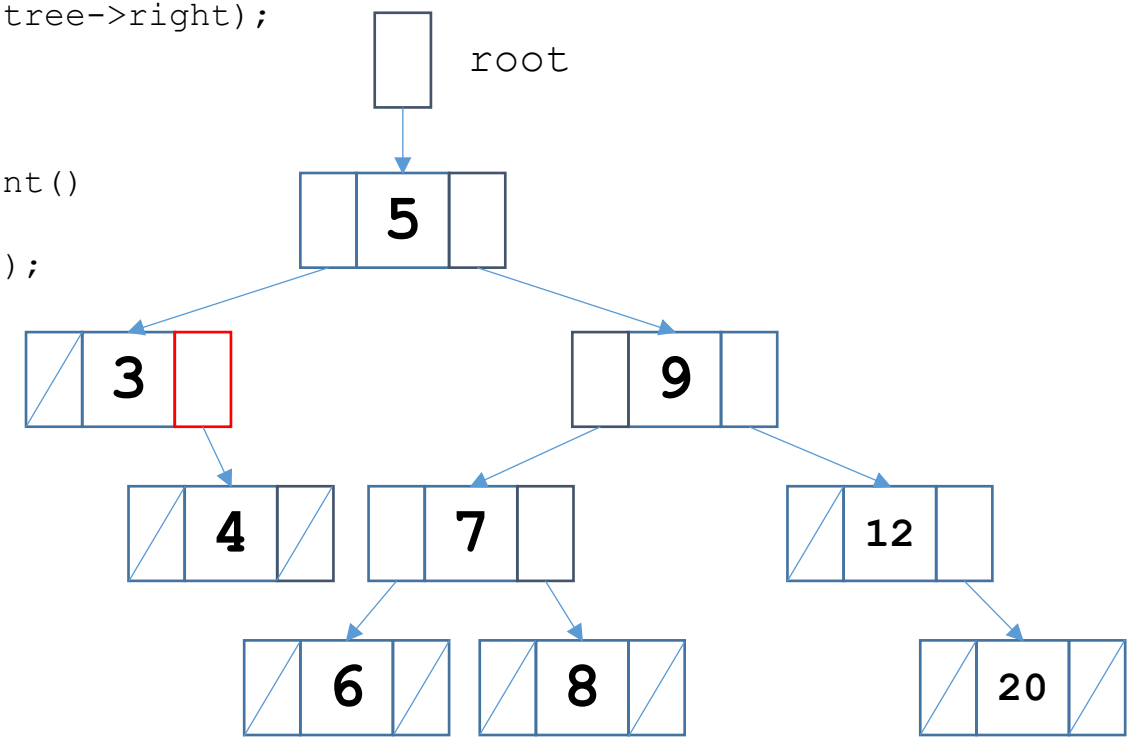
3 4

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



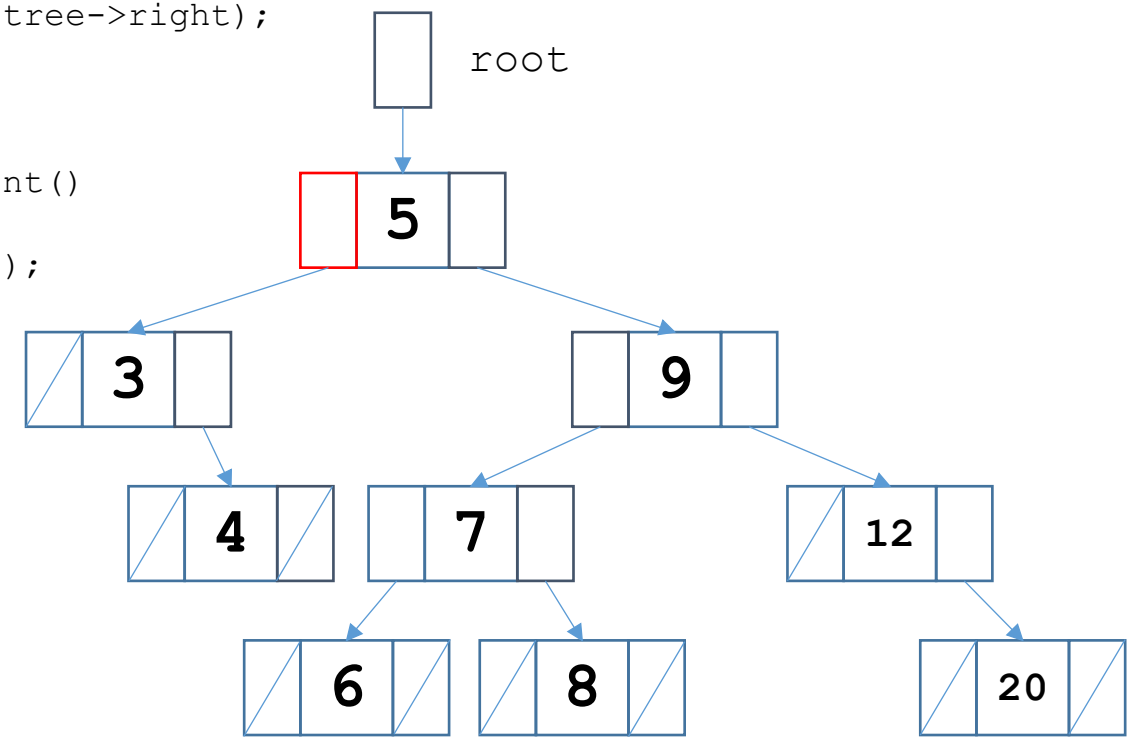
3 4

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



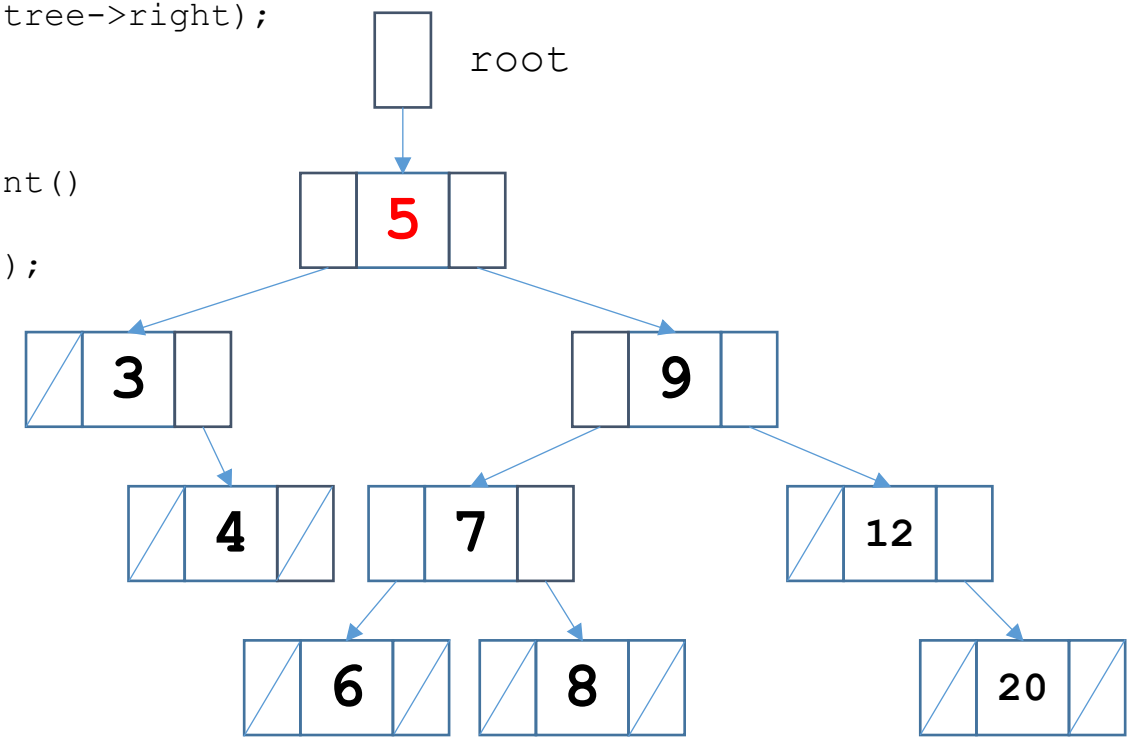
3 4

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



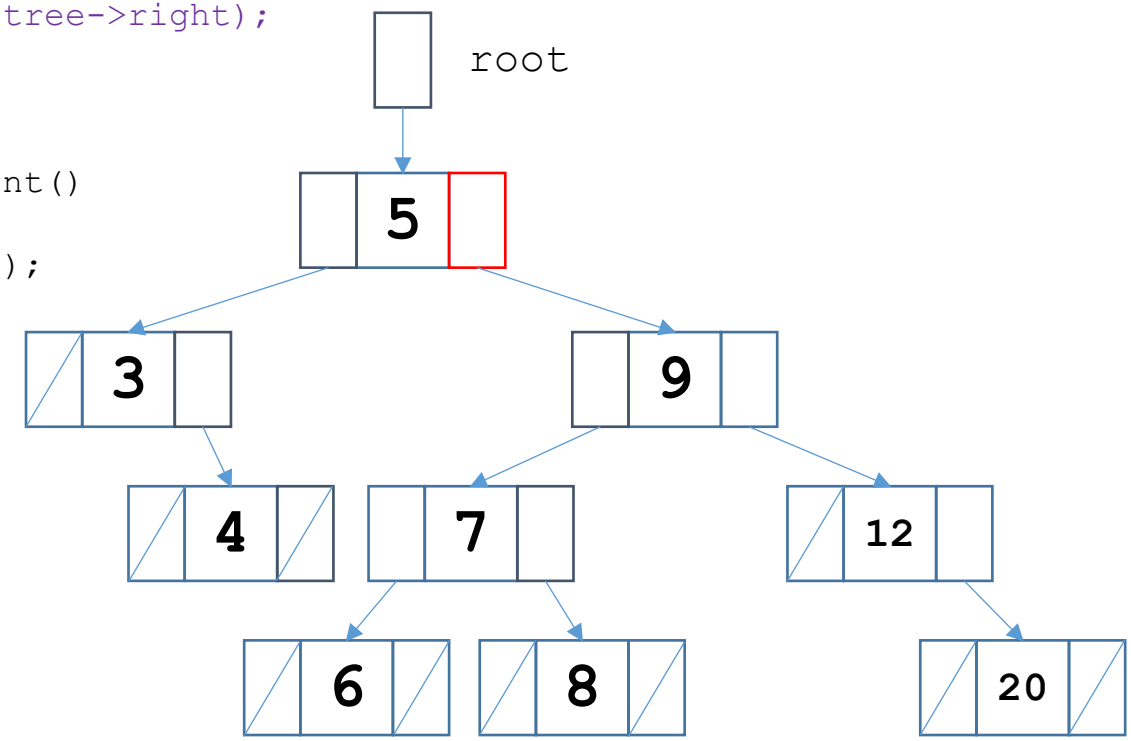
3 4

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



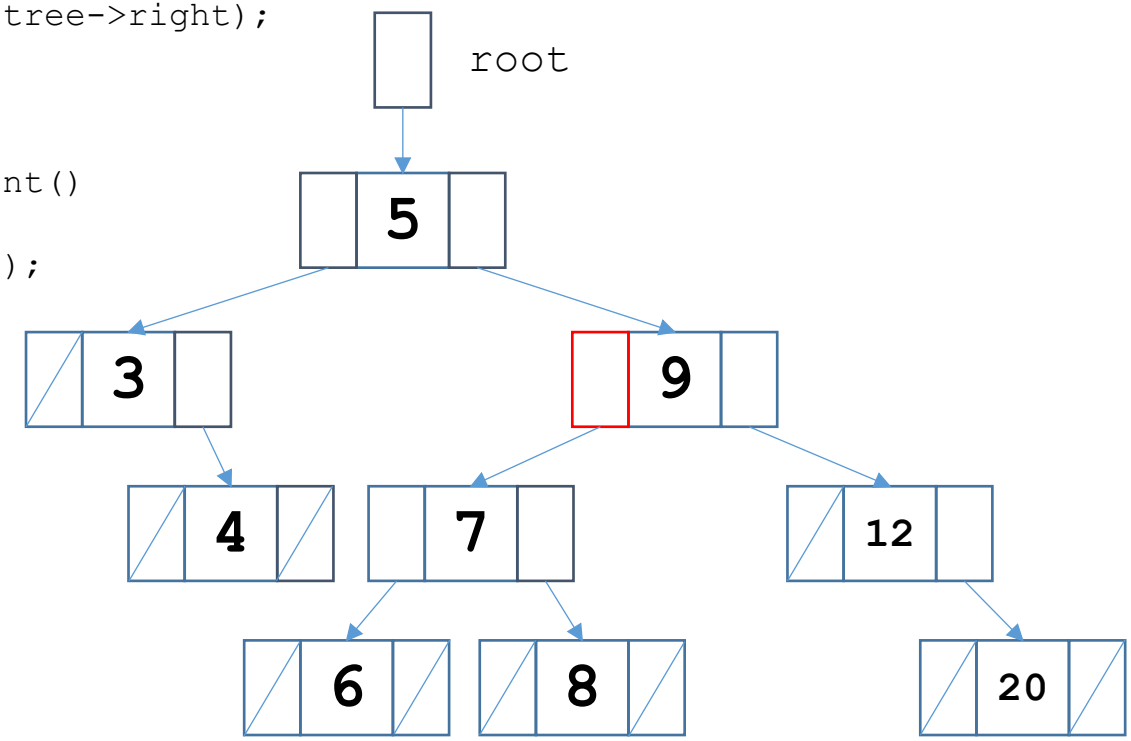
3 4 5

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



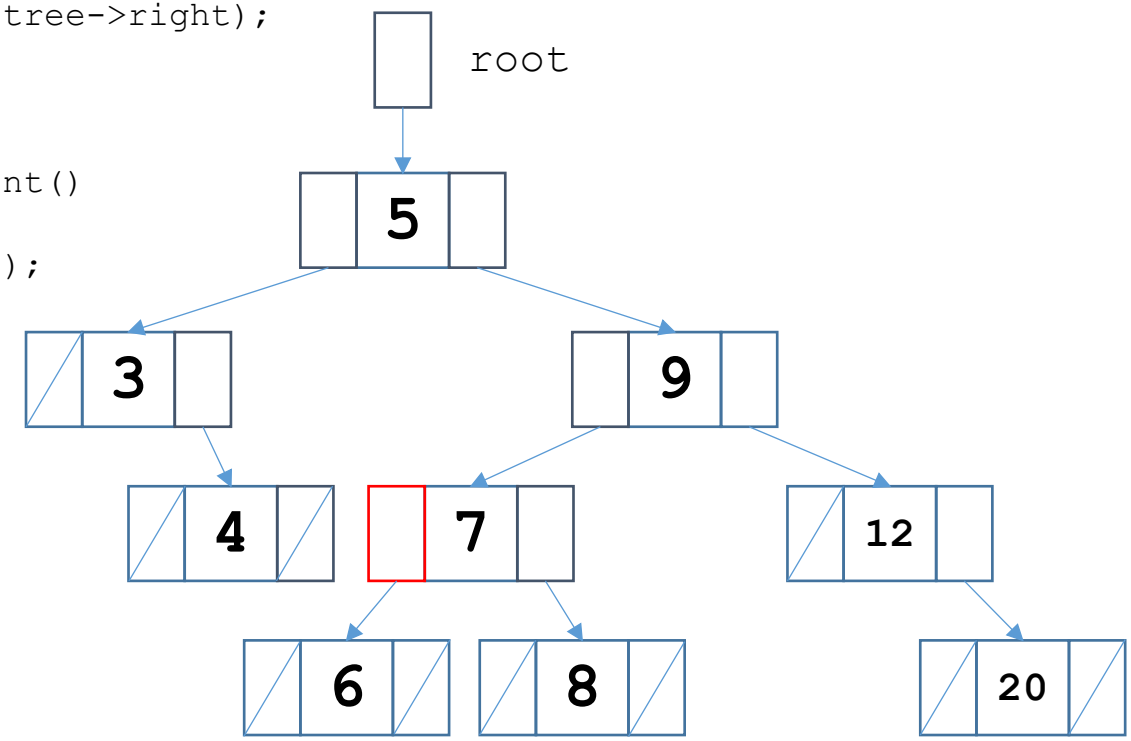
3 4 5

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



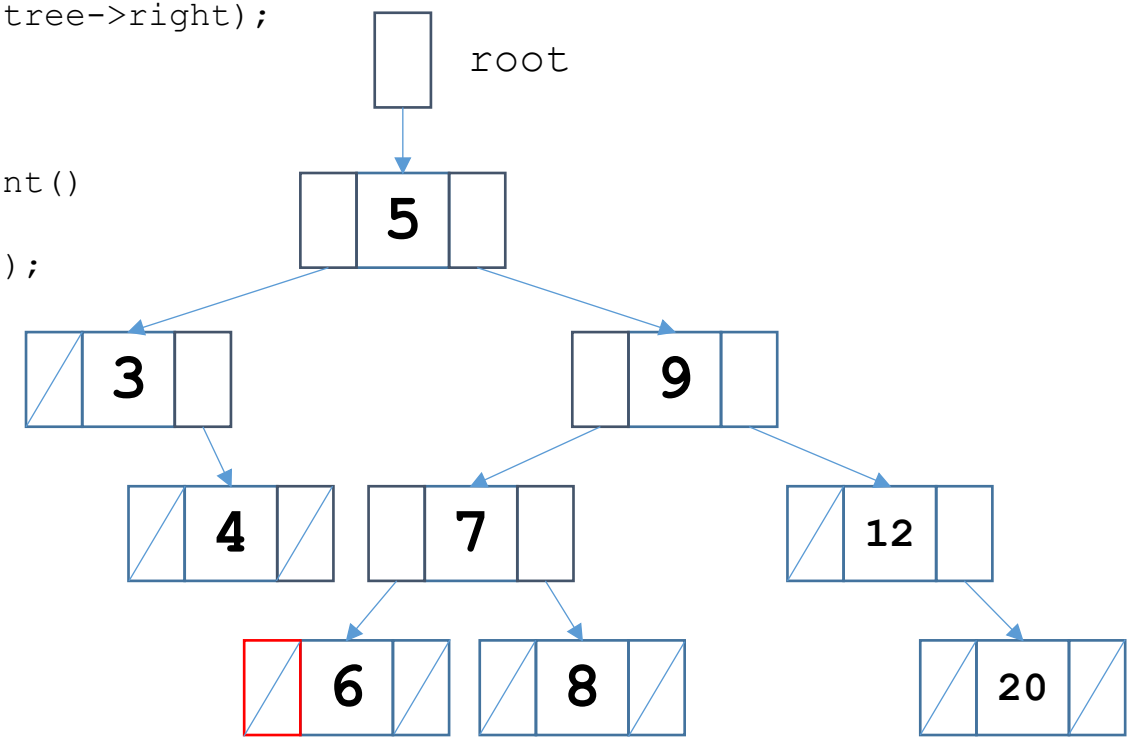
3 4 5

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



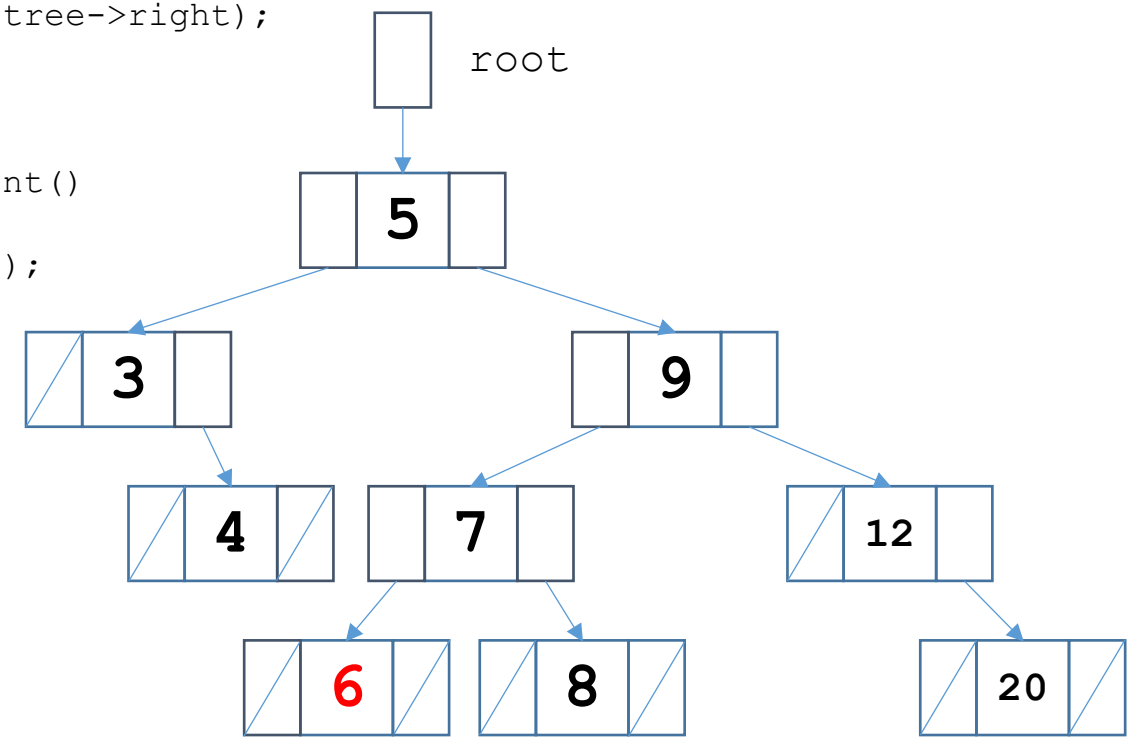
3 4 5

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



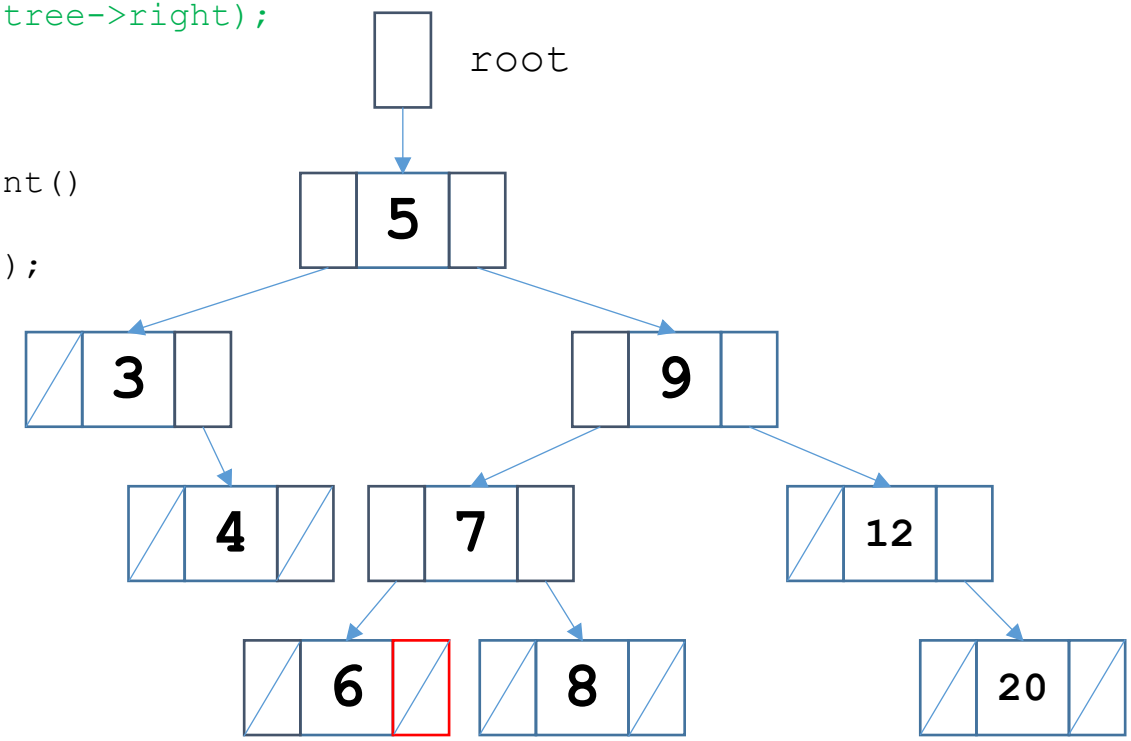
3 4 5

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



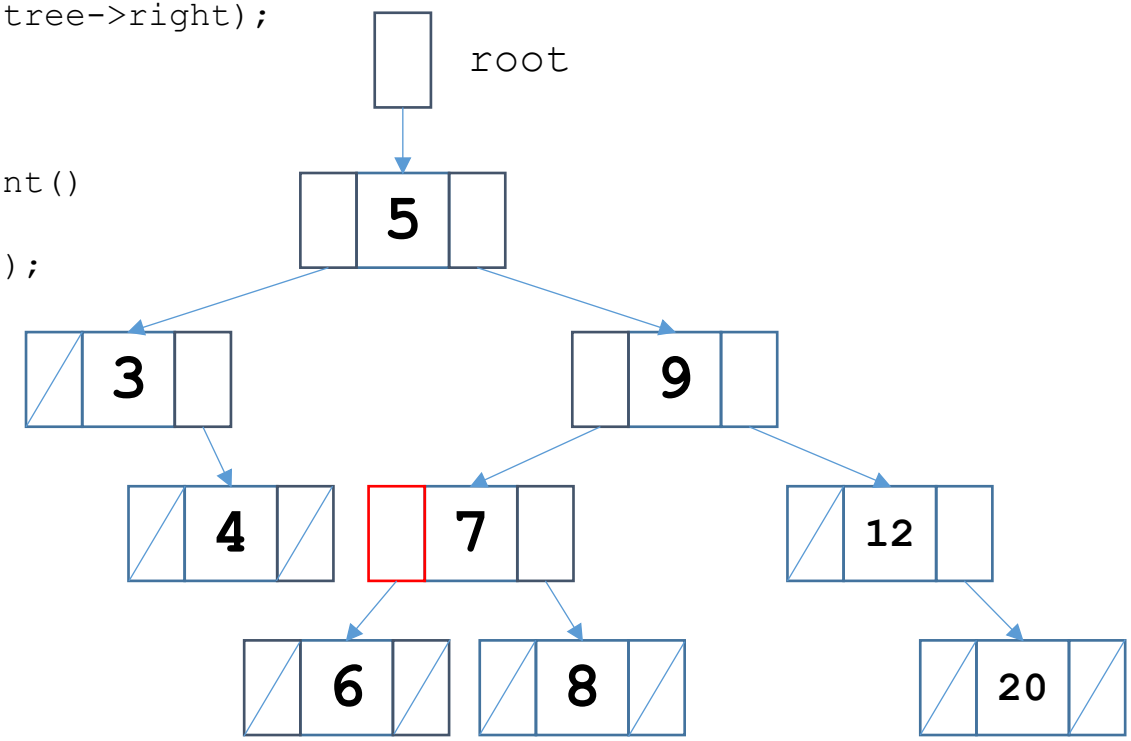
3 4 5 6

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



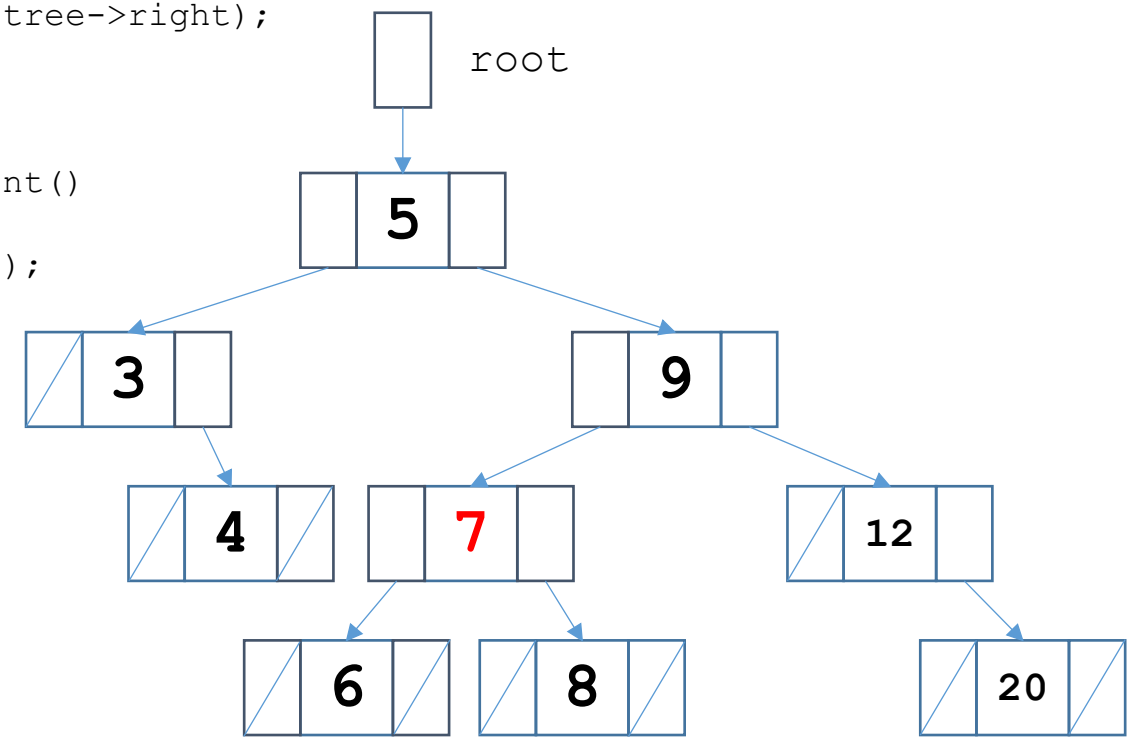
3 4 5 6

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



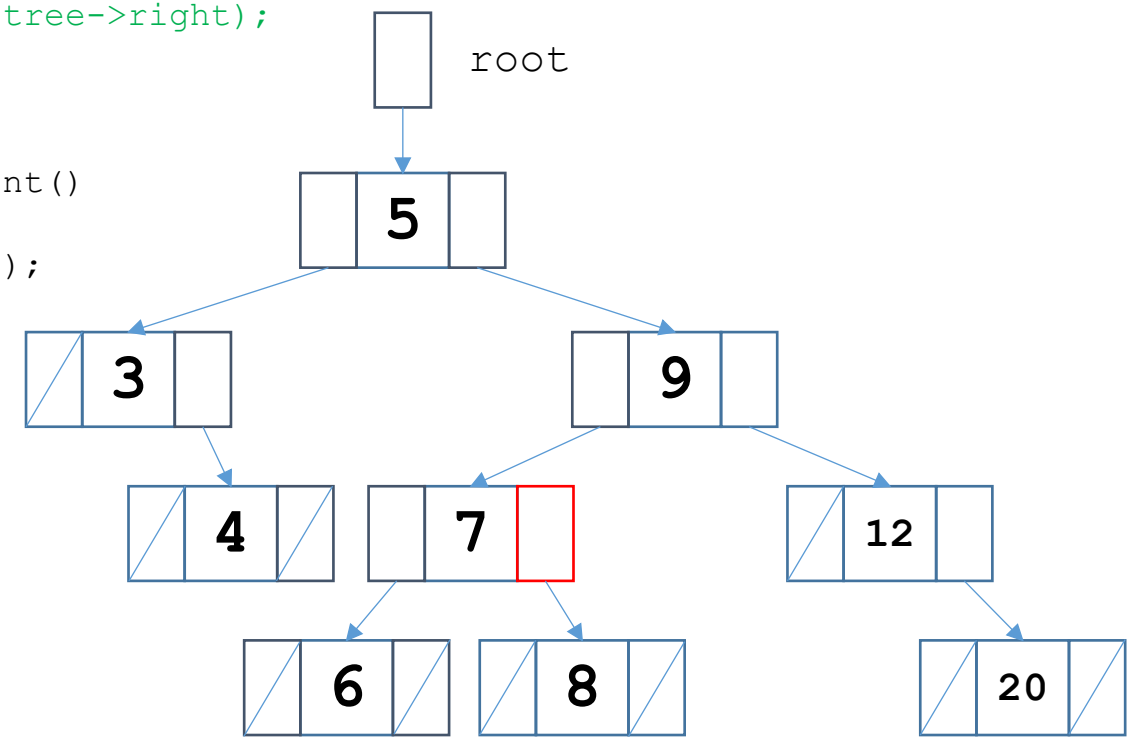
3 4 5 6

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



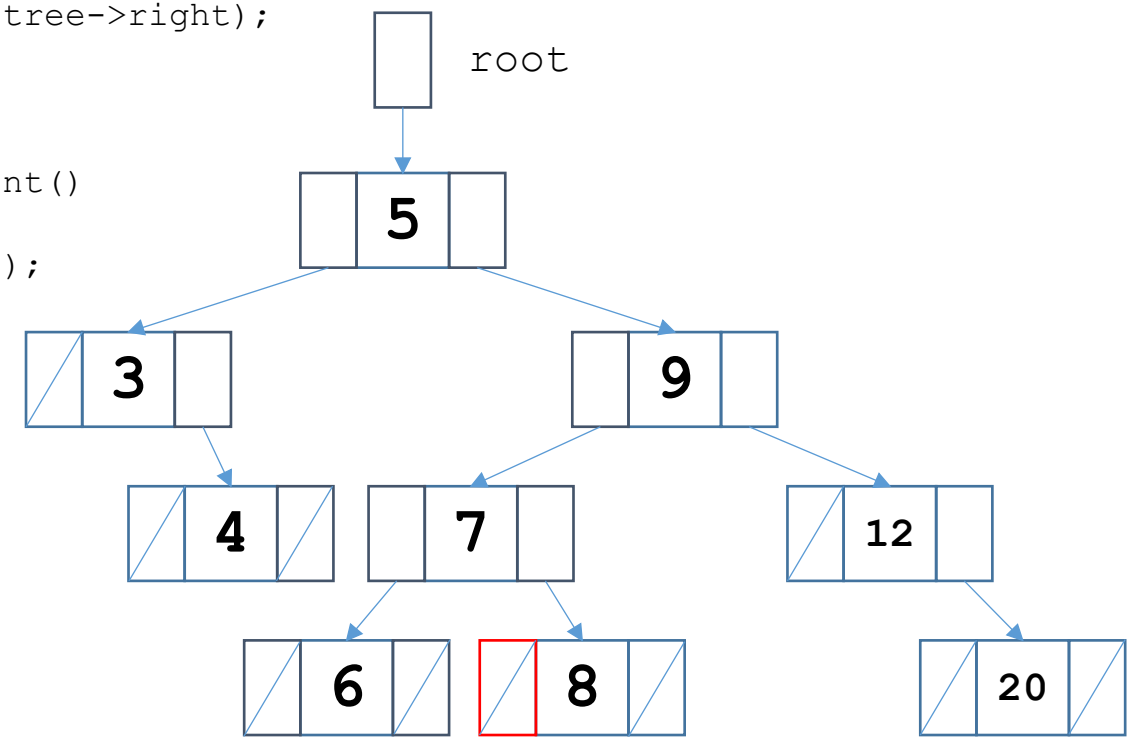
3 4 5 6 7

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



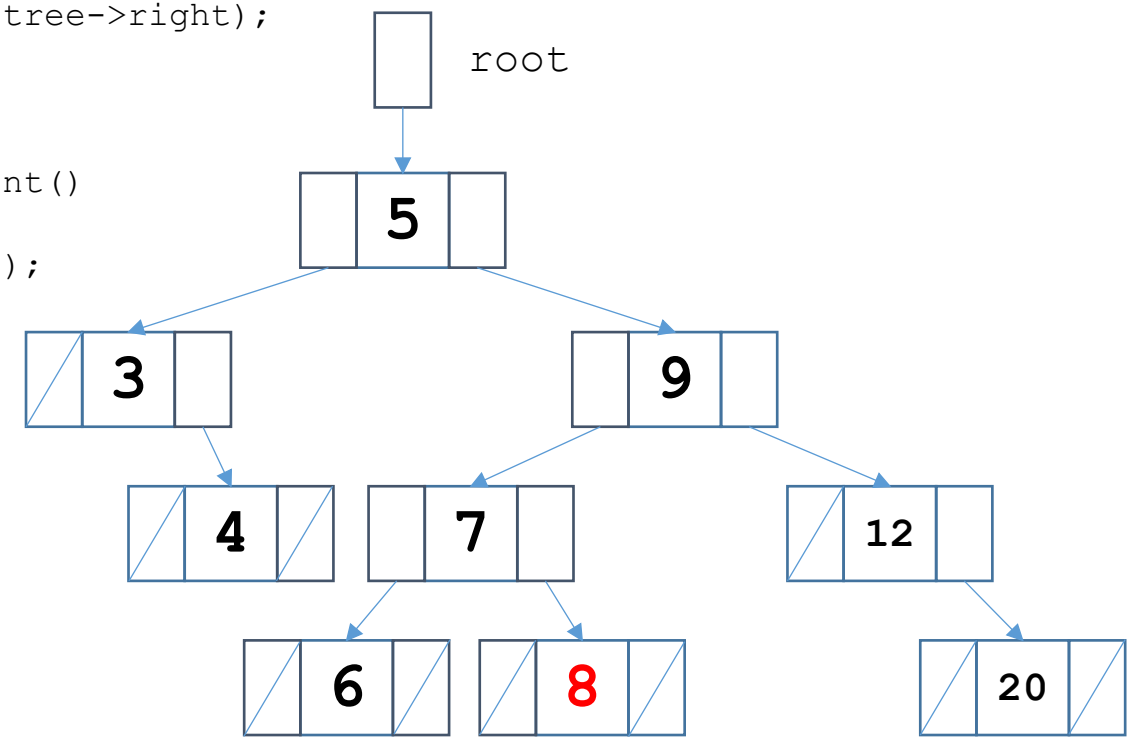
3 4 5 6 7

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



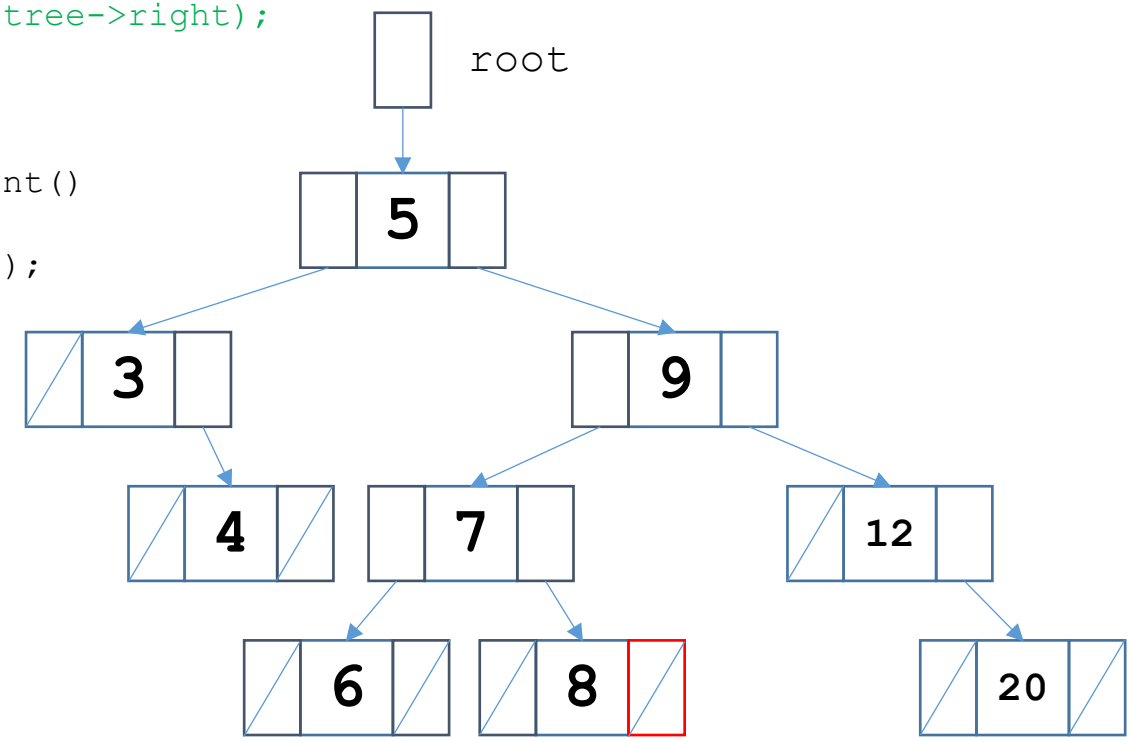
3 4 5 6 7

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



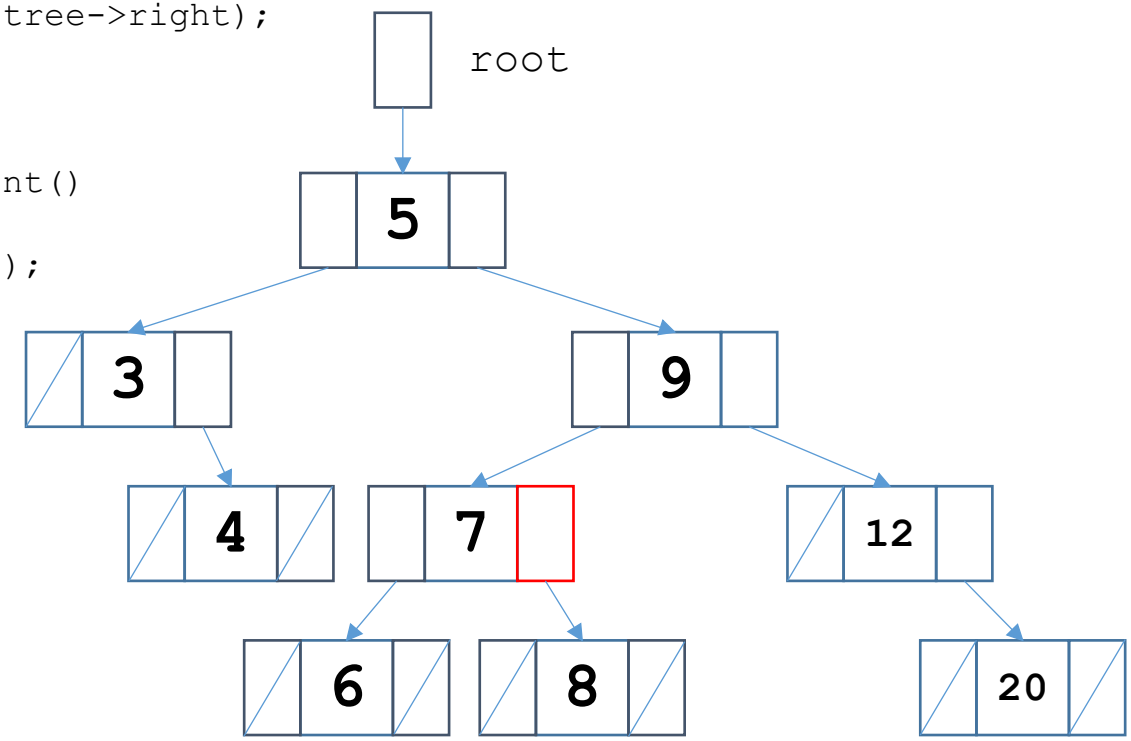
3 4 5 6 7 8

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



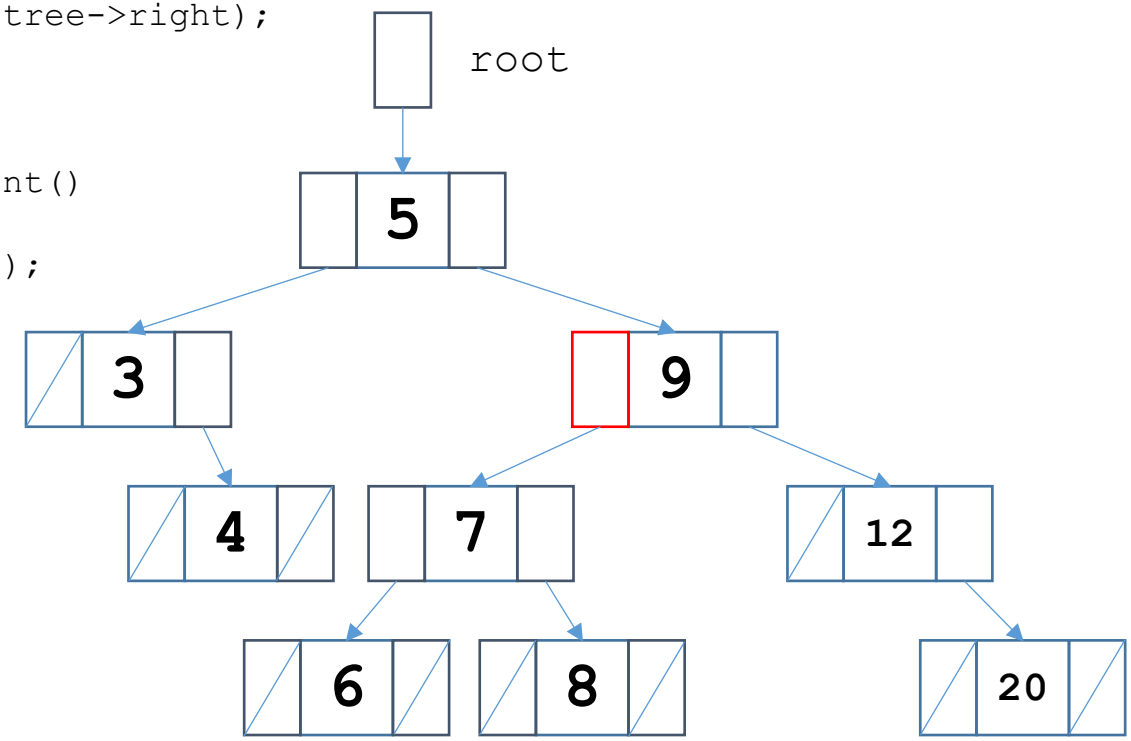
3 4 5 6 7 8

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



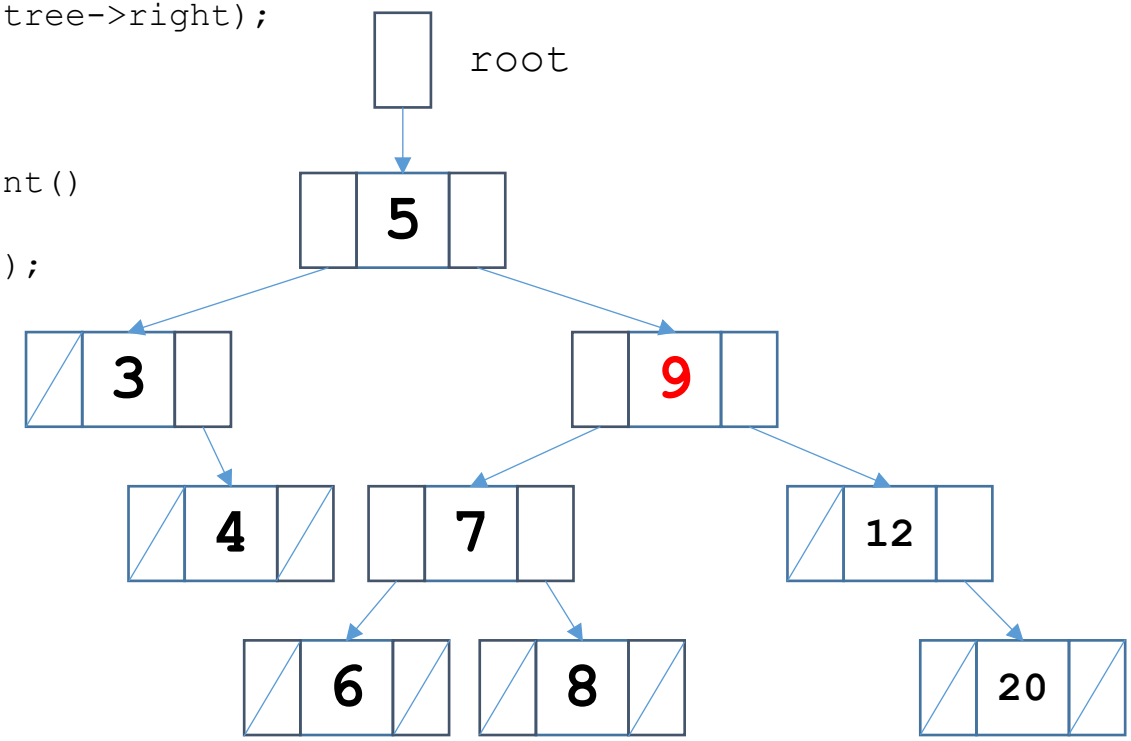
3 4 5 6 7 8

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



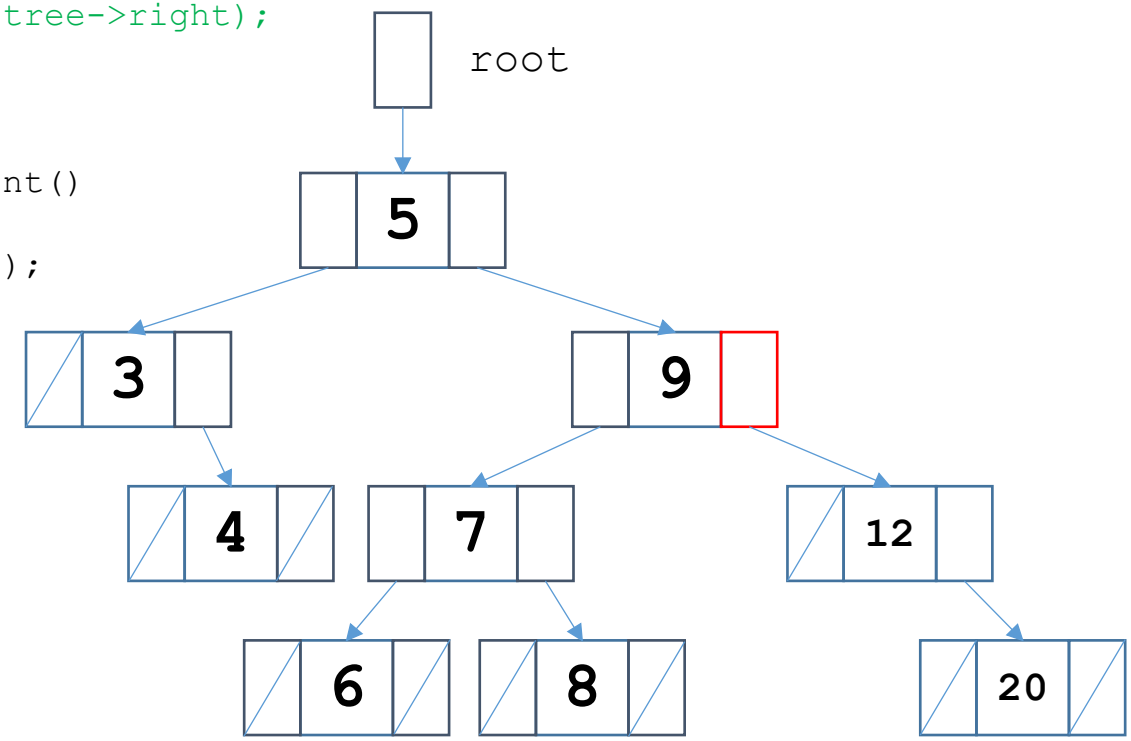
3 4 5 6 7 8

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



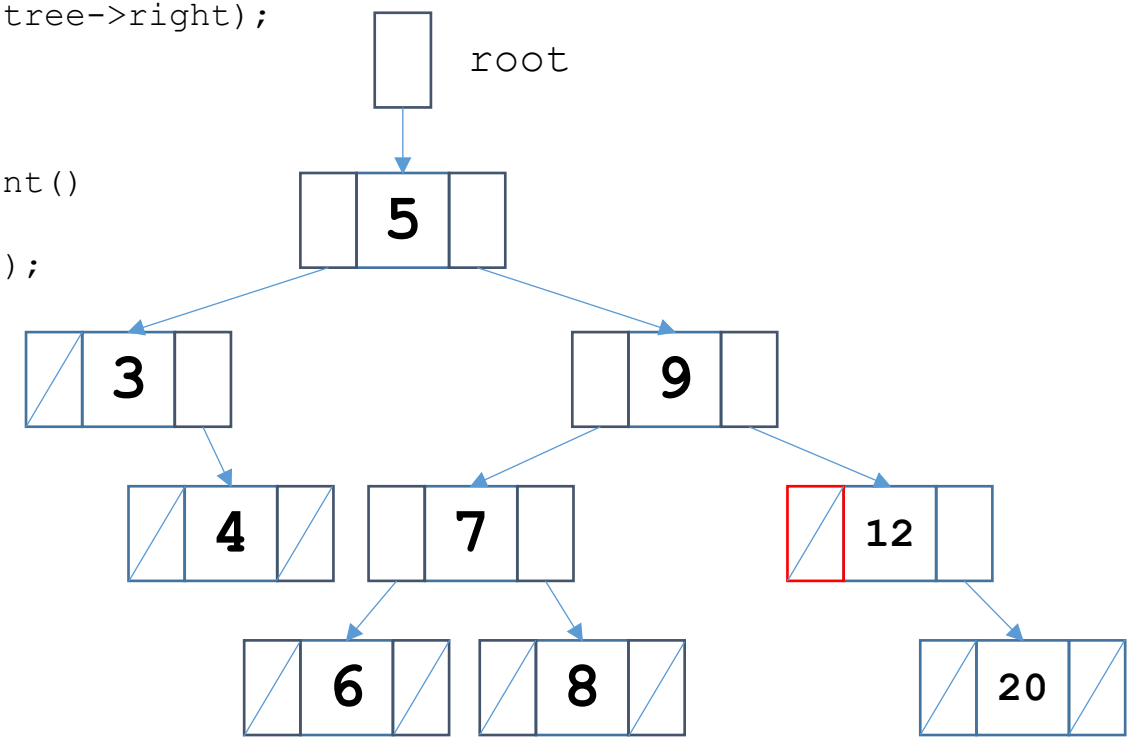
3 4 5 6 7 8 9

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



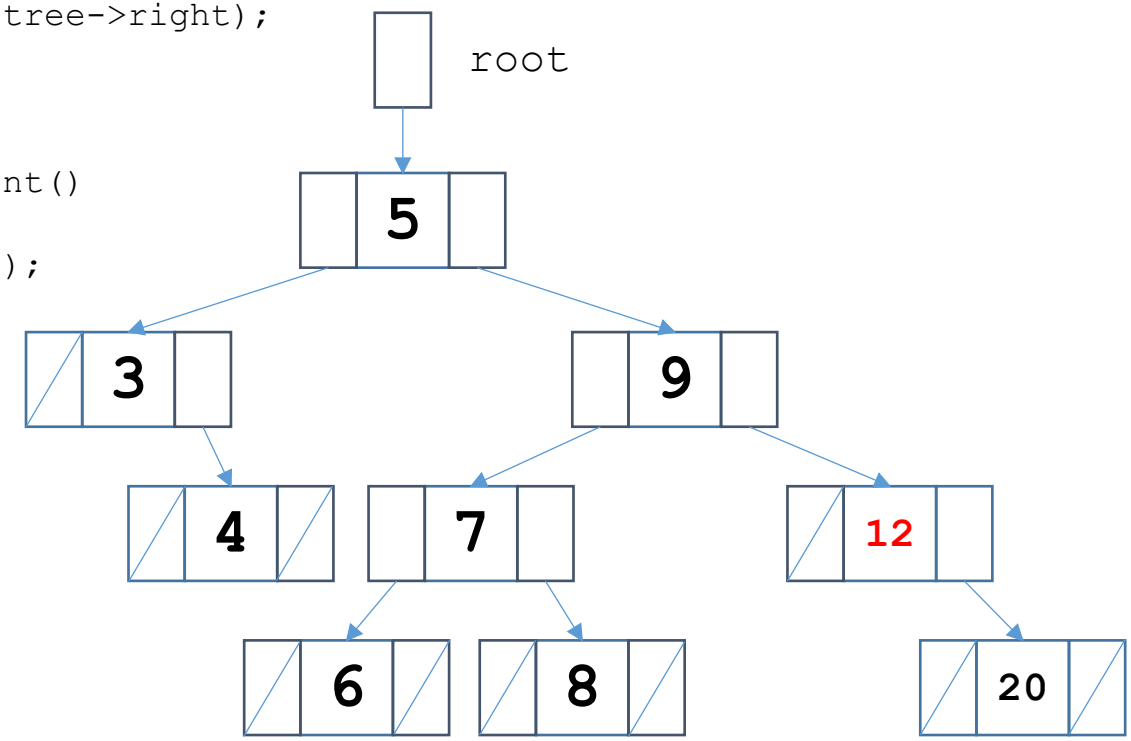
3 4 5 6 7 8 9

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



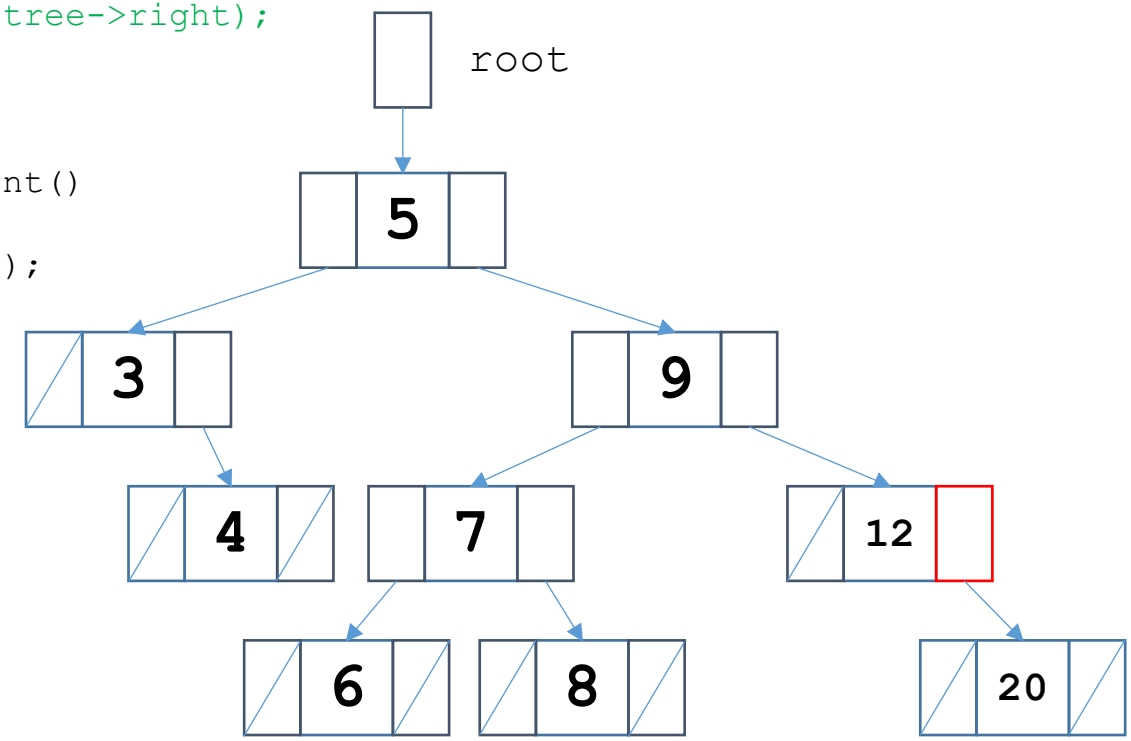
3 4 5 6 7 8 9

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



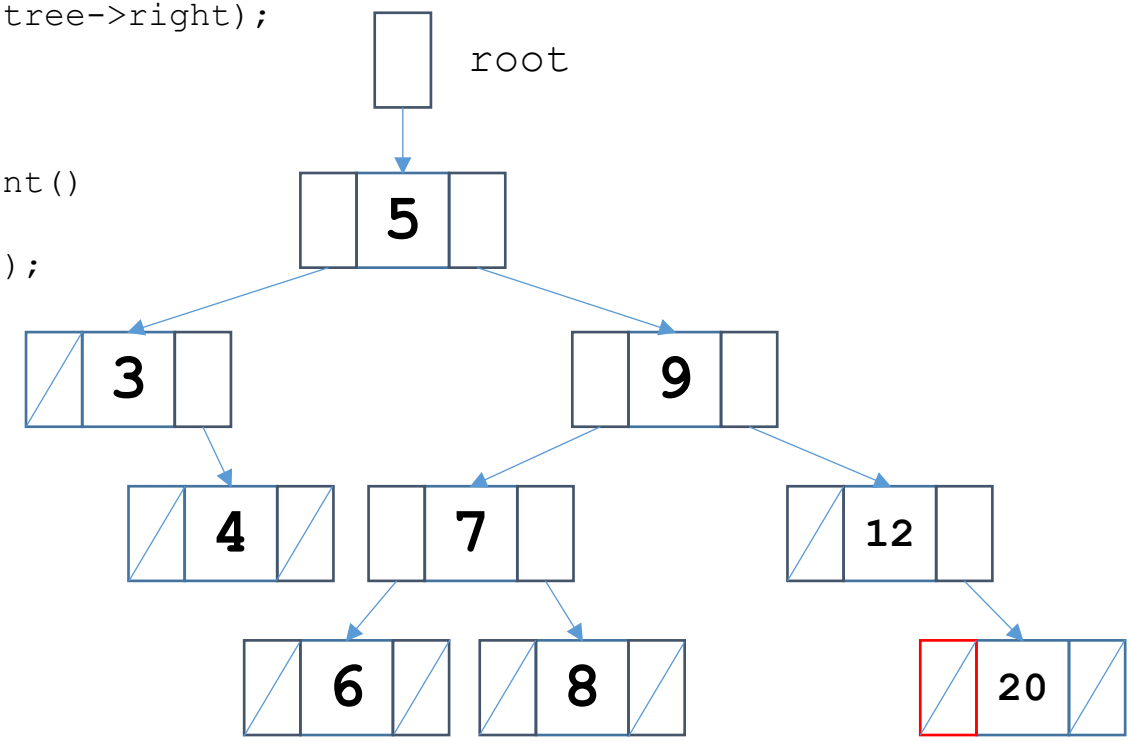
3 4 5 6 7 8 9 12

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



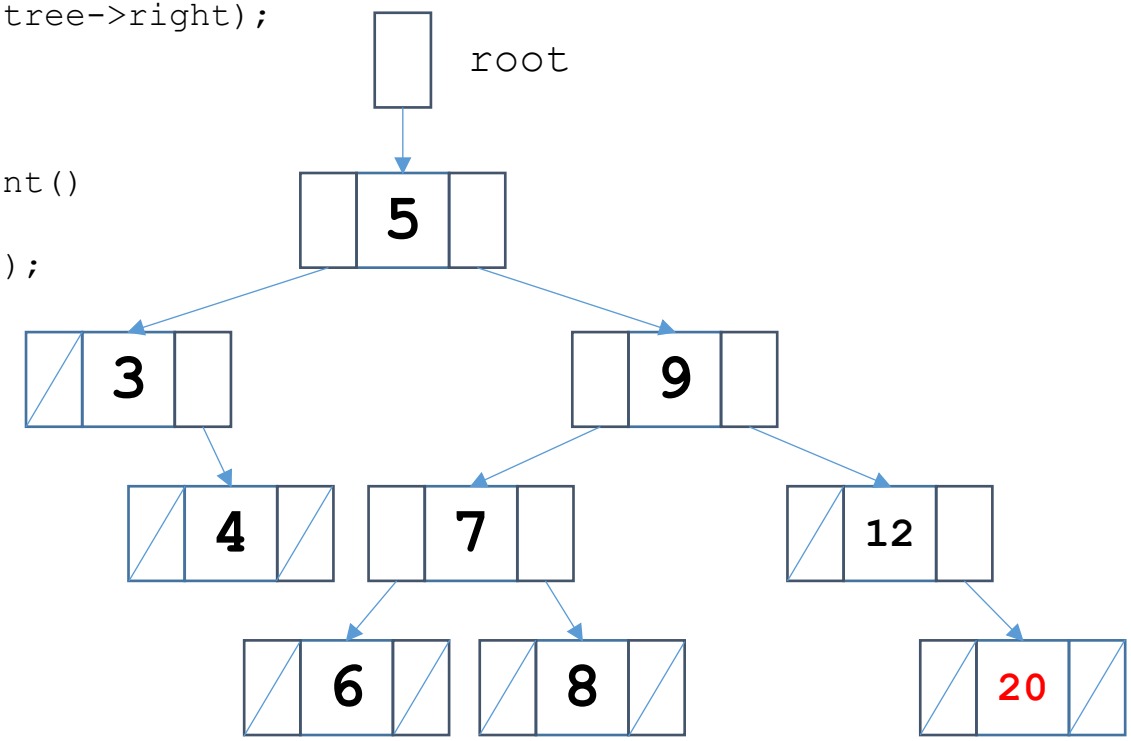
3 4 5 6 7 8 9 12

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



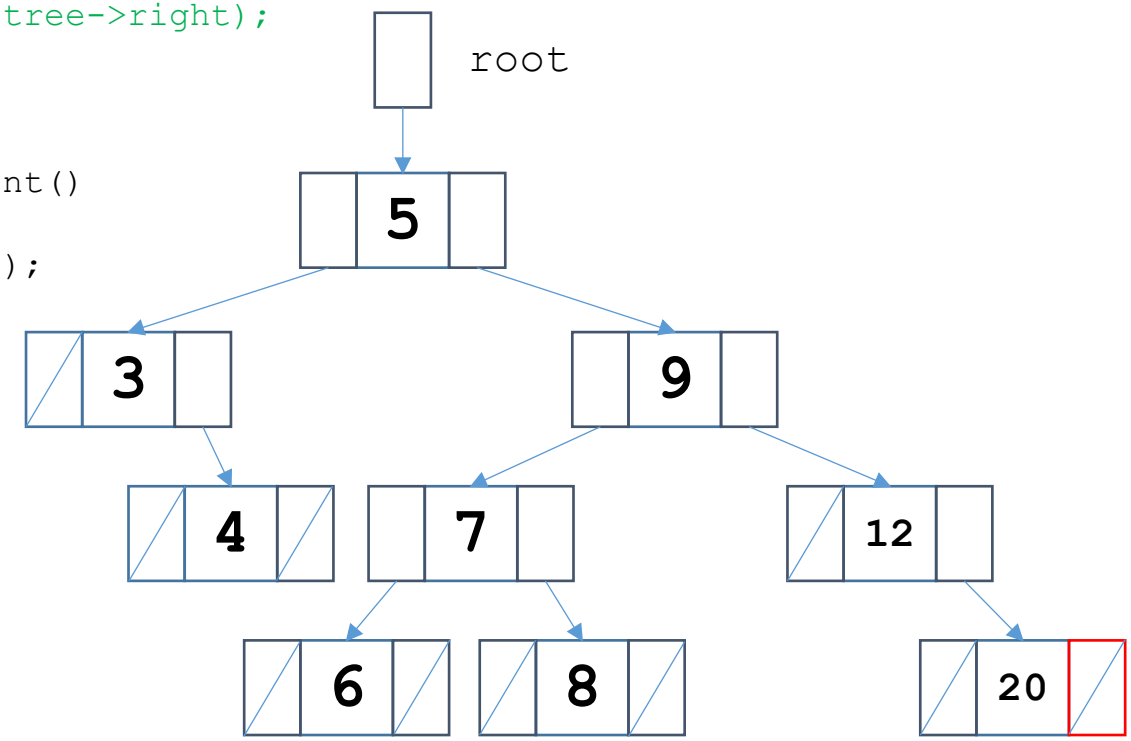
3 4 5 6 7 8 9 12

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



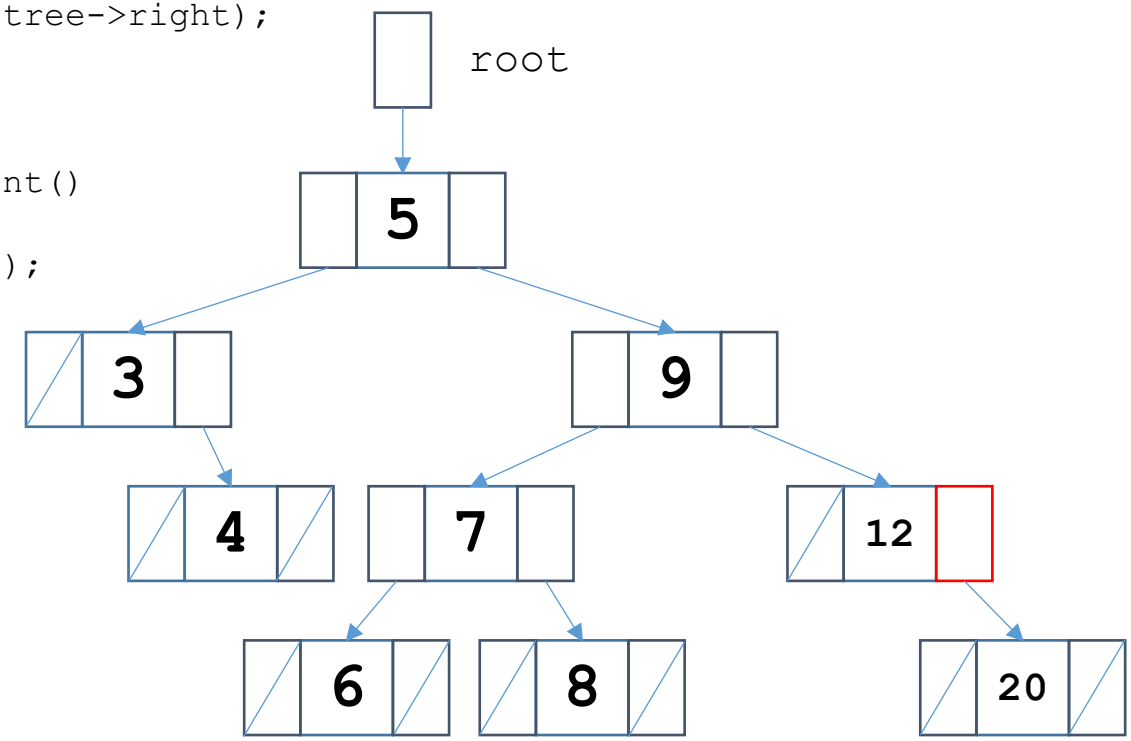
3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



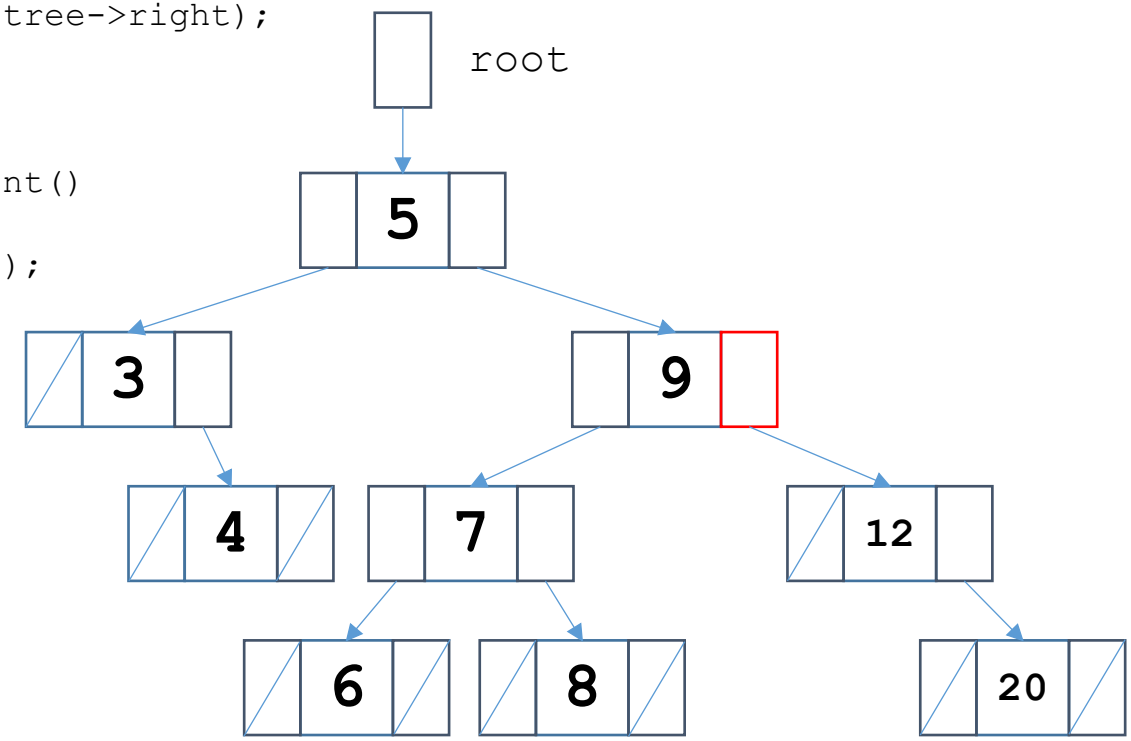
3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



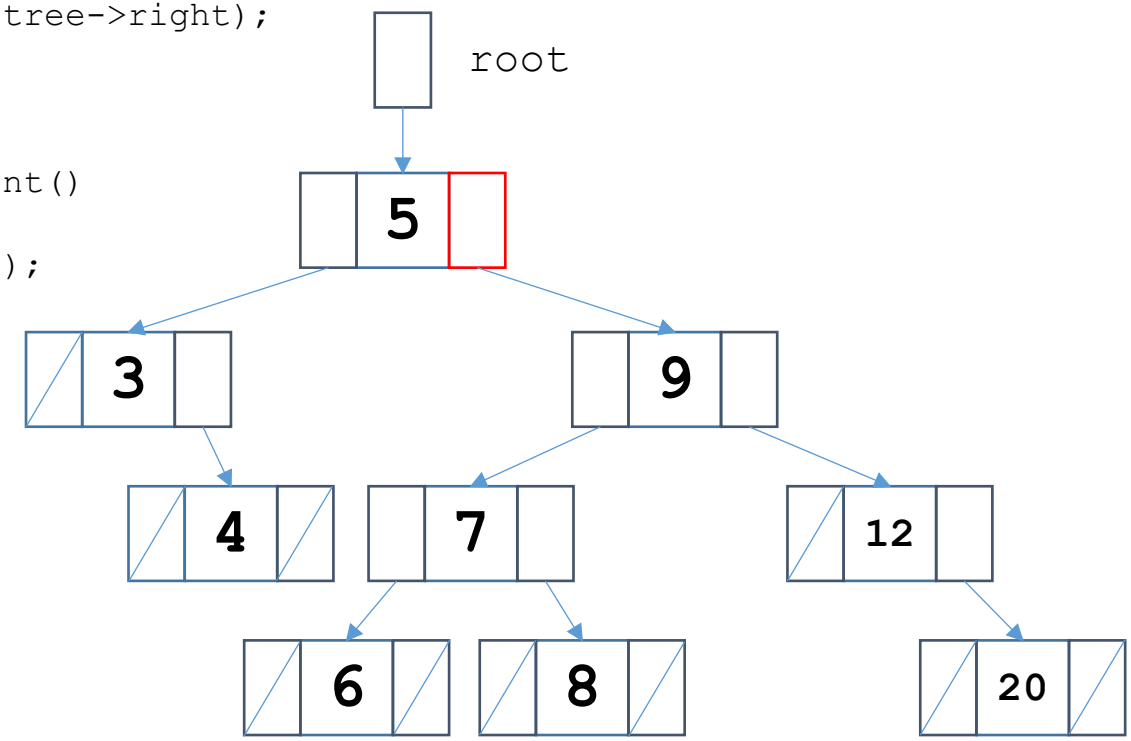
3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



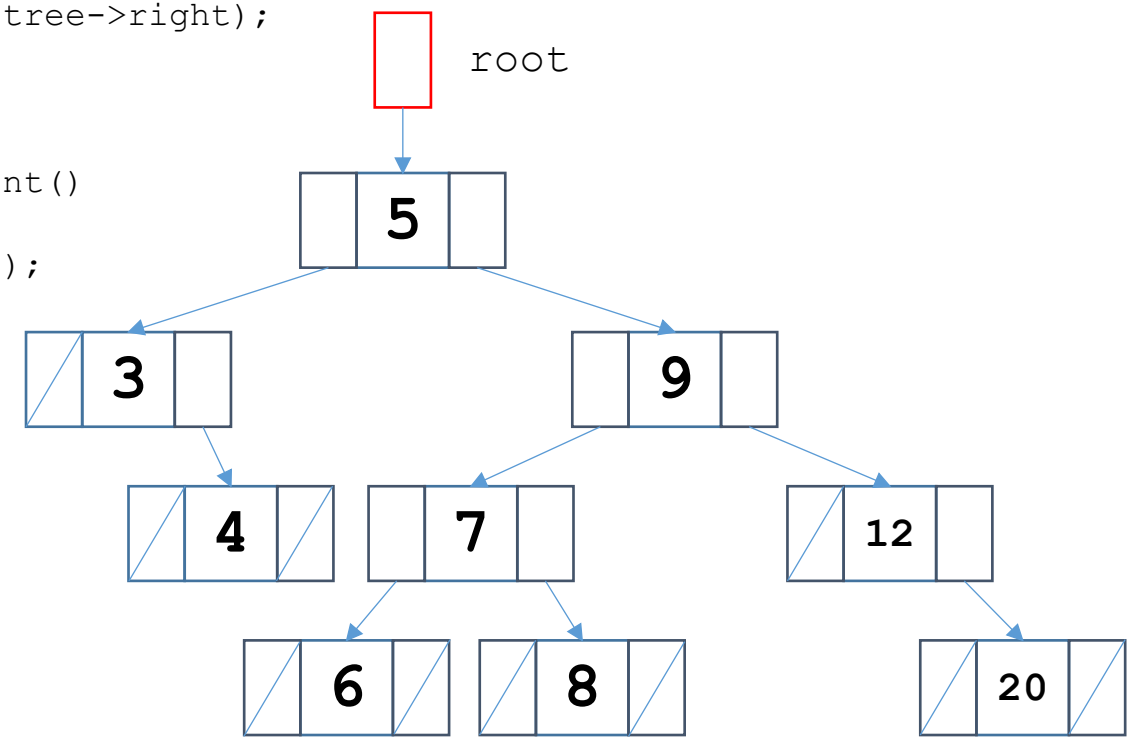
3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



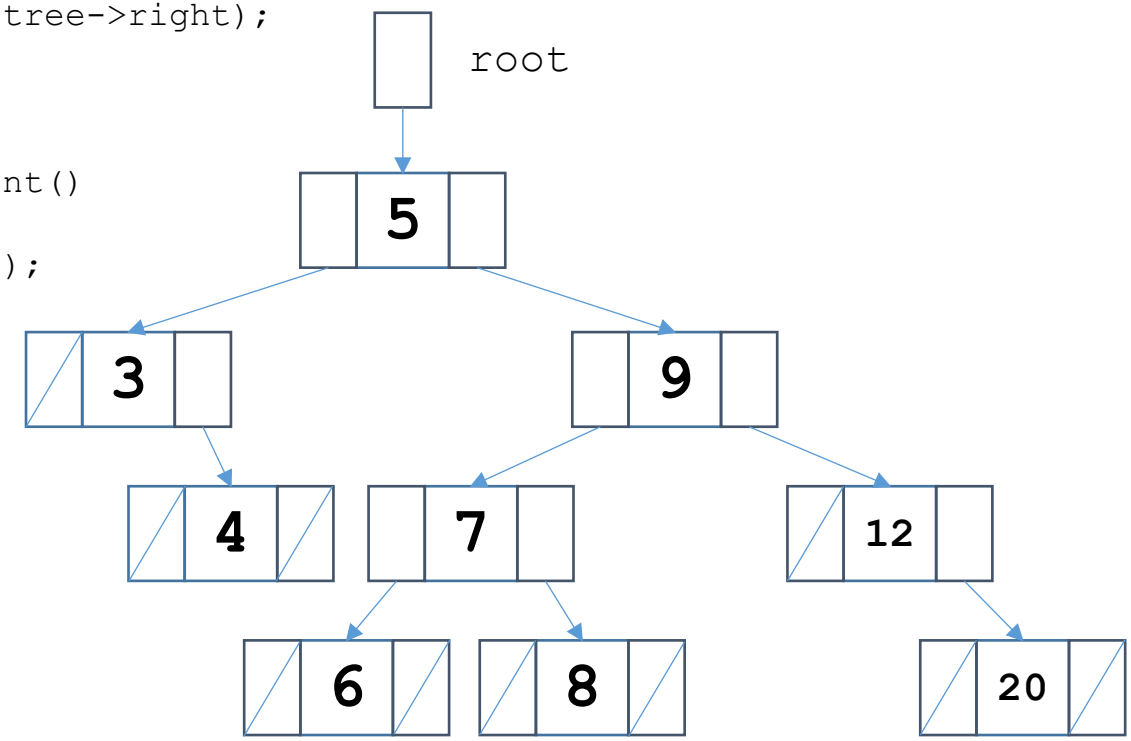
3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



3 4 5 6 7 8 9 12 20

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info <<
endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```



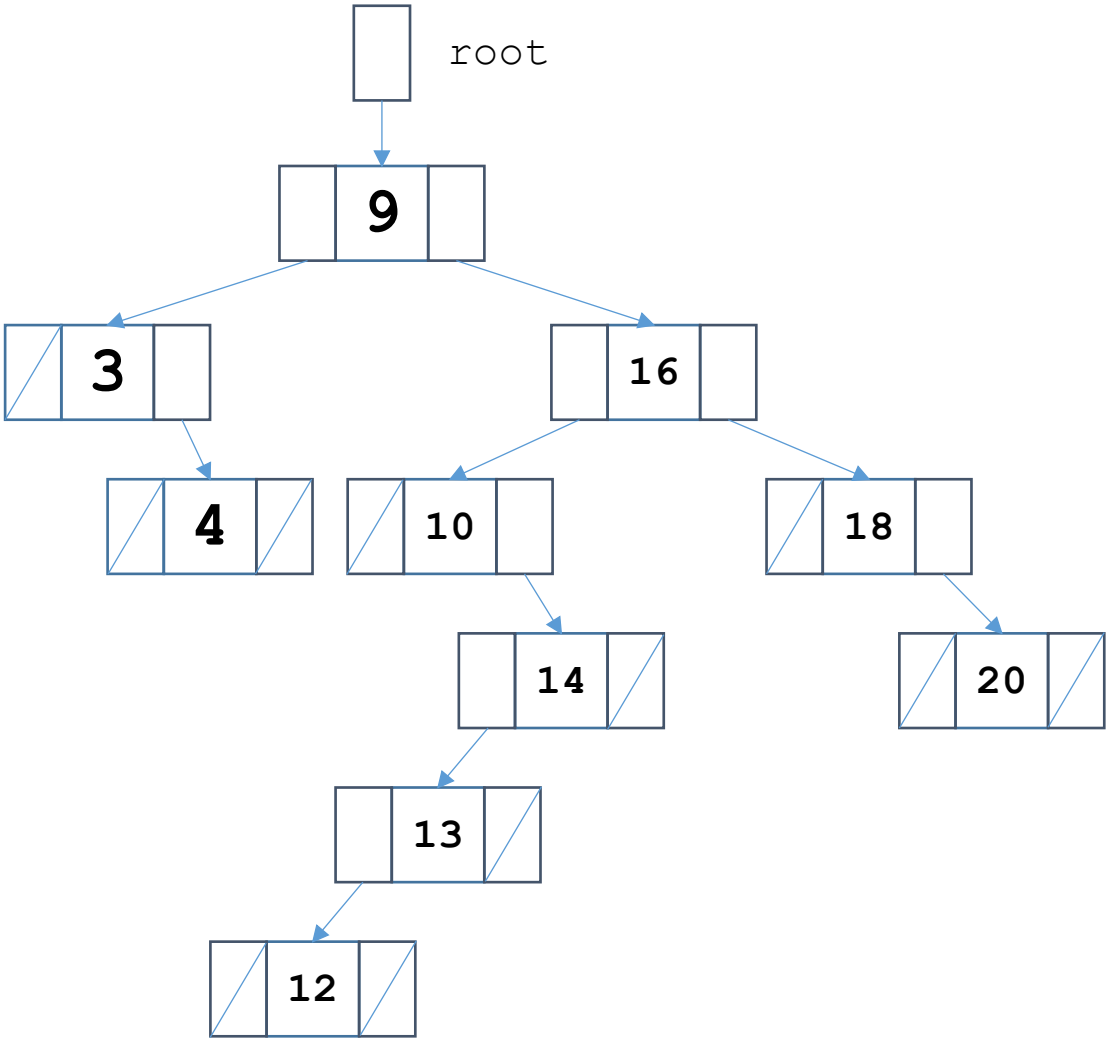
3 4 5 6 7 8 9 12 20

Function `Print`

```
void PrintTree(TreeNode* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        std::cout << tree->info << std::endl;
        PrintTree(tree->right);
    }
}

void TreeType::Print()
{
    PrintTree(root);
}
```

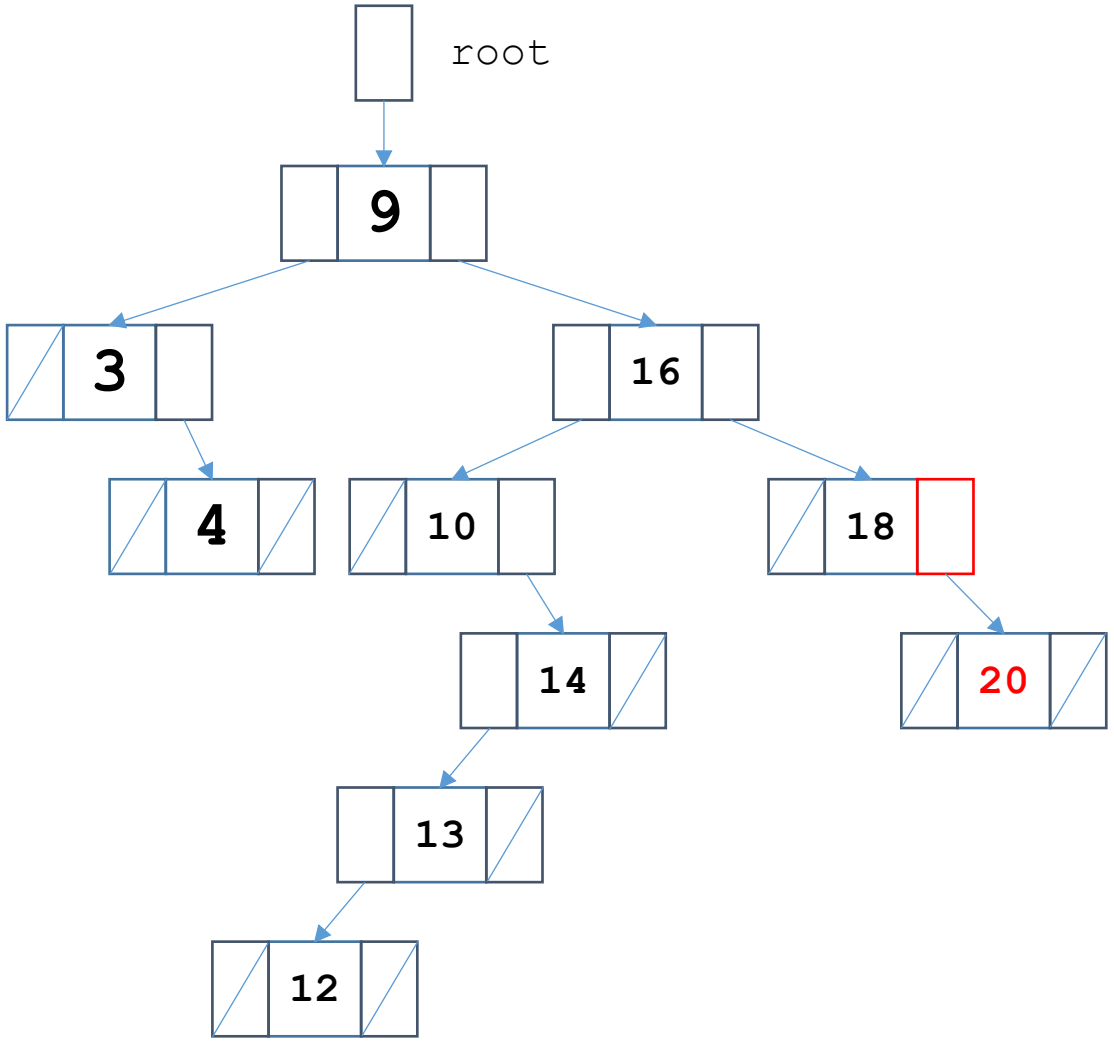
O(N)

Deleting a Node



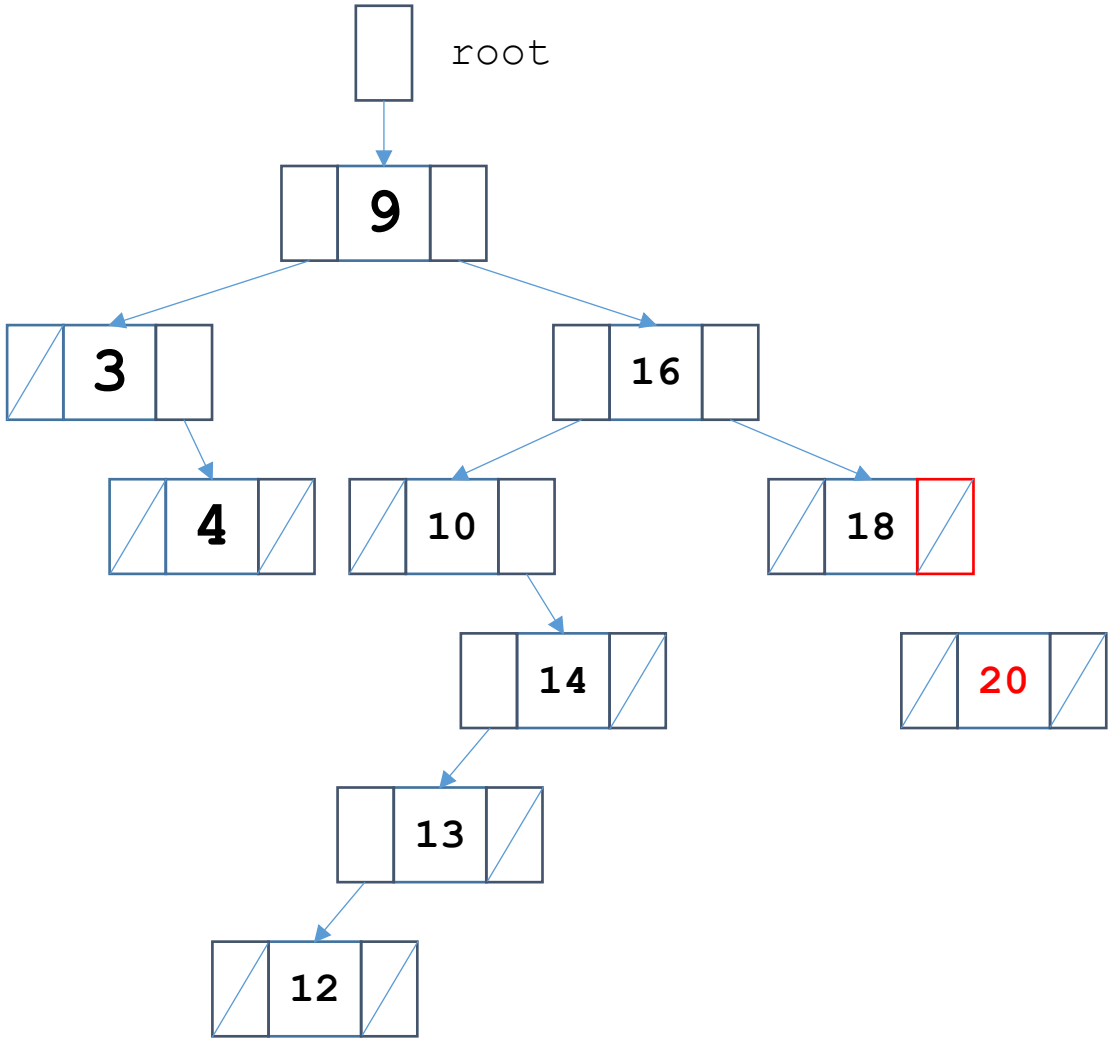
Deleting a Node

Deleting a leaf node (20).



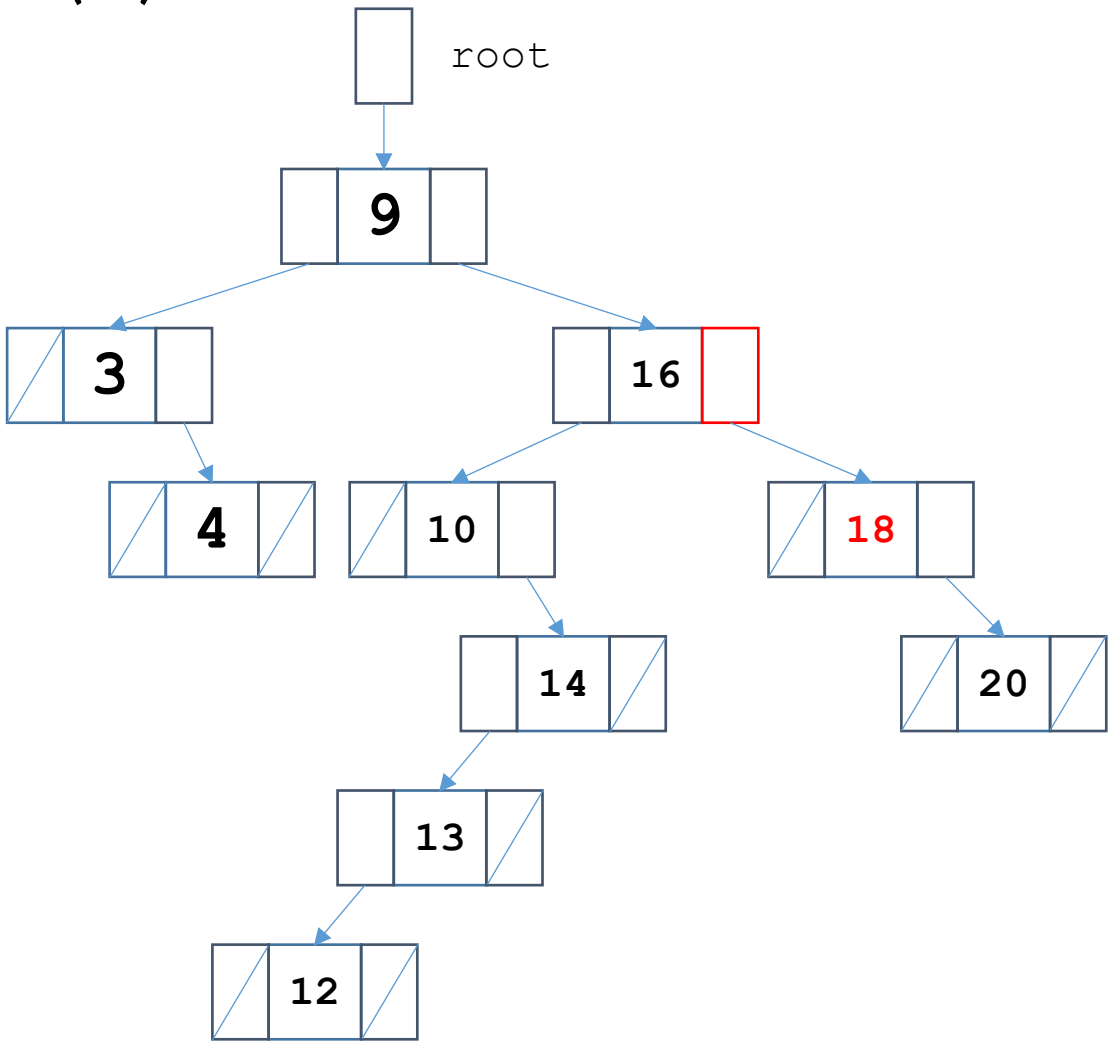
Deleting a Node

Deleting a leaf node (20).



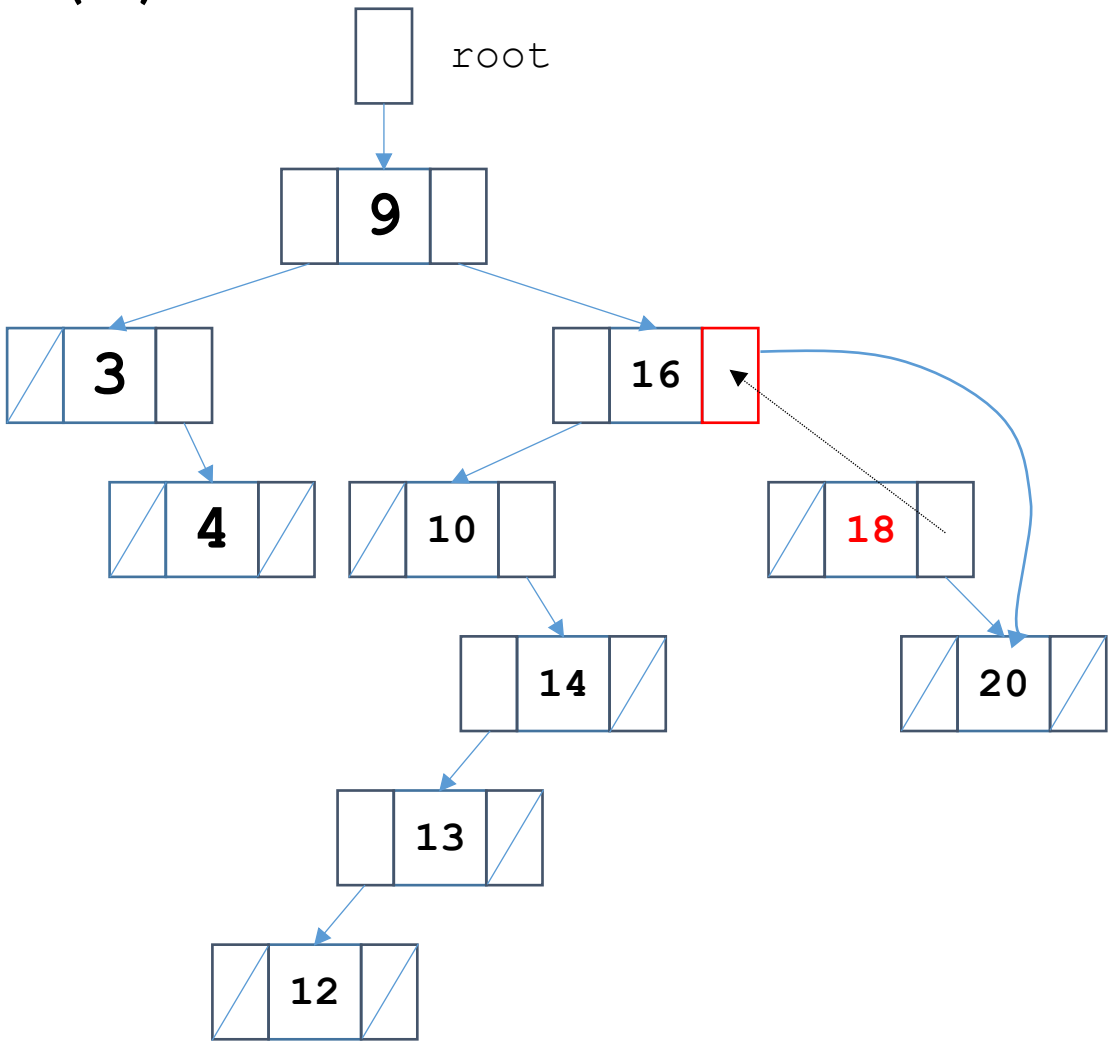
Deleting a Node

Deleting a node with one child (18).



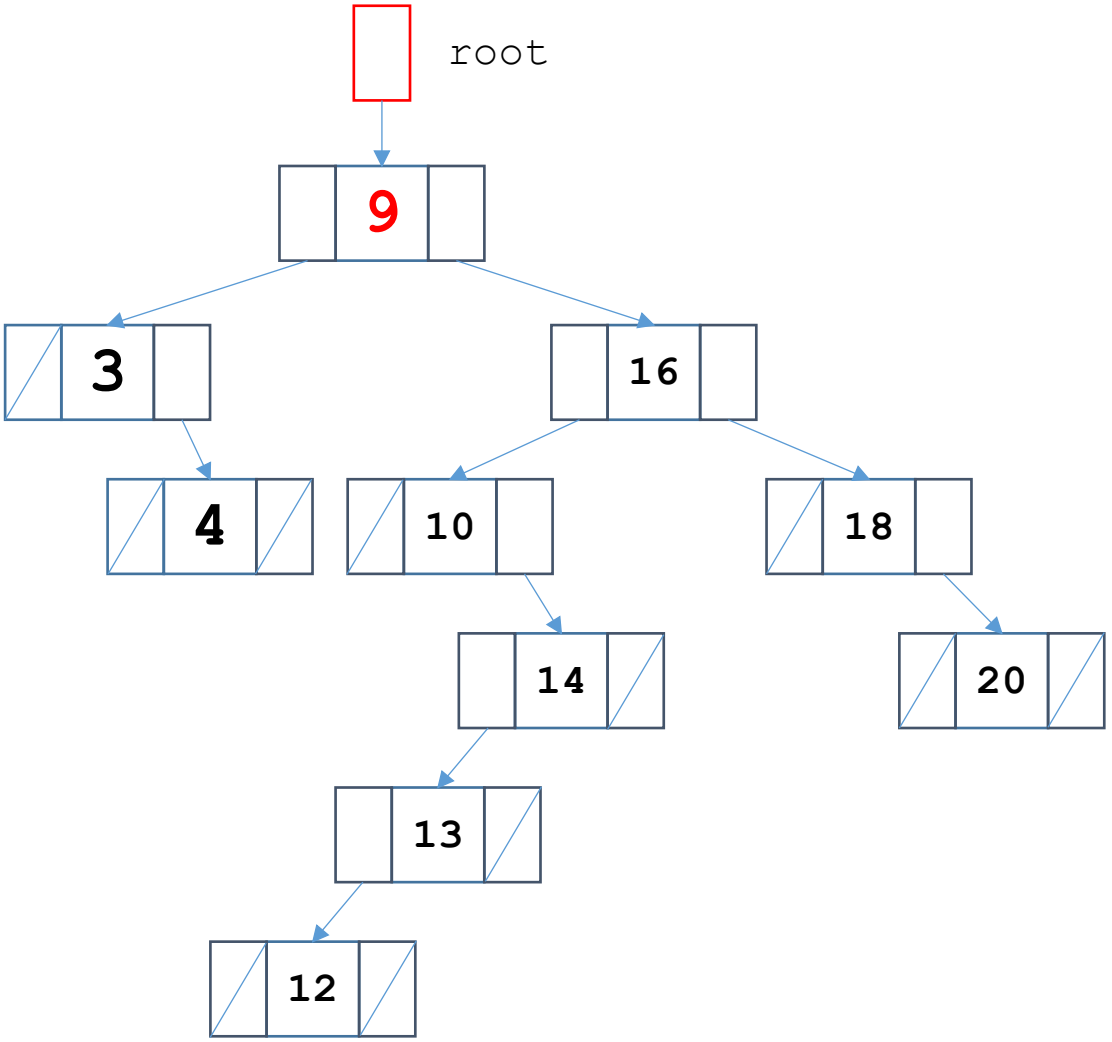
Deleting a Node

Deleting a node with one child (18).



Deleting a Node

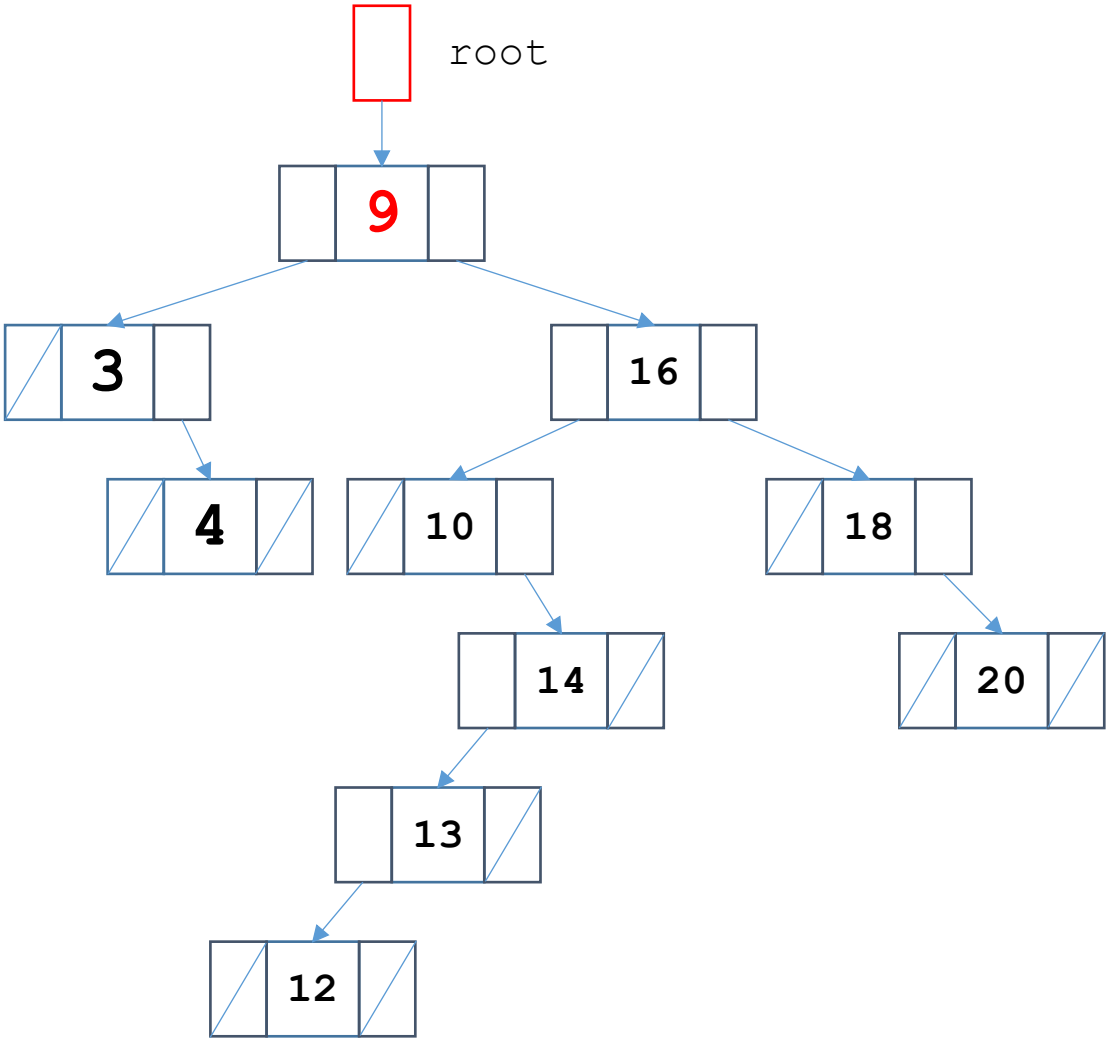
Deleting a node with two children (9).



Deleting a Node

Deleting a node with two children (9).

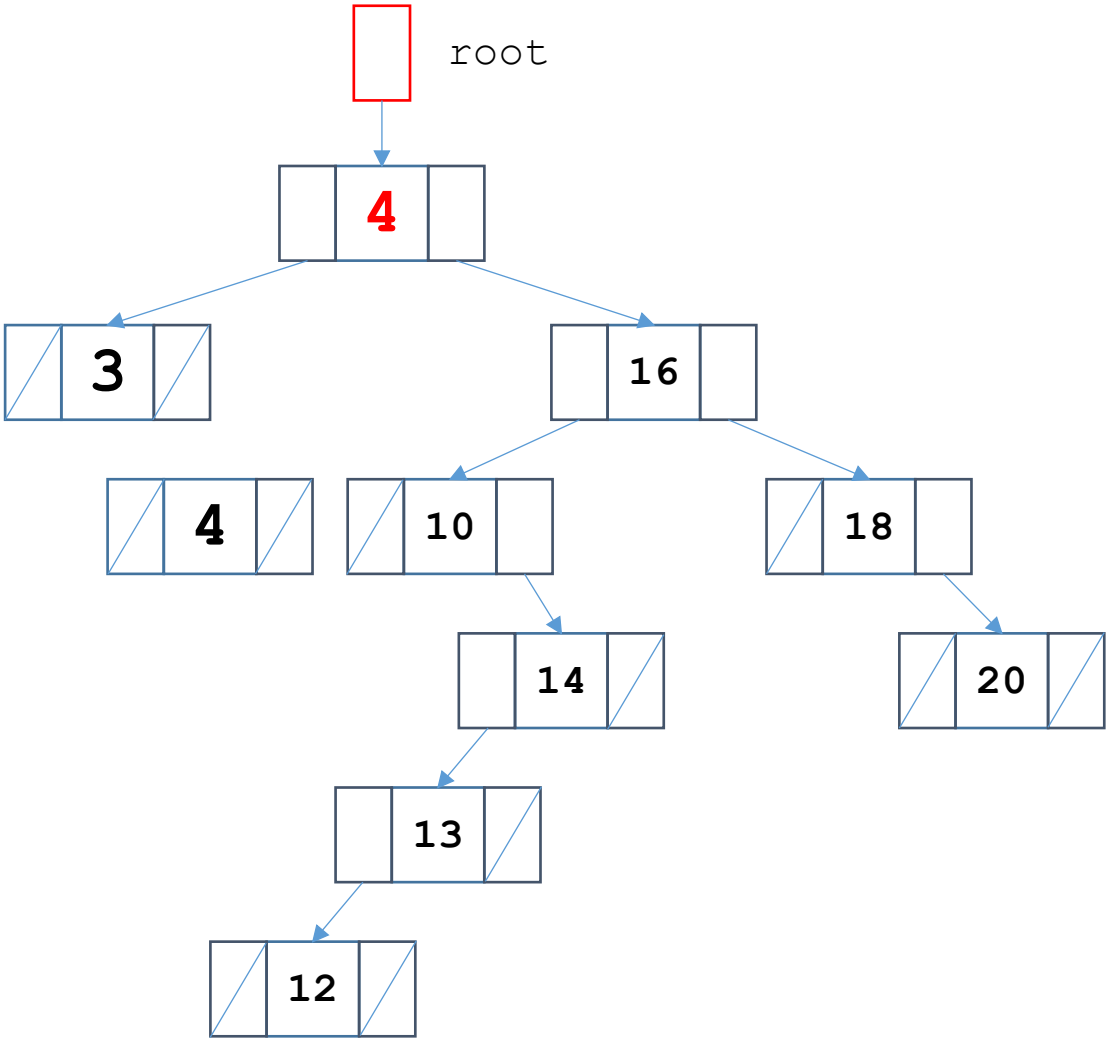
Replace it with its predecessor (**rightmost nodes in the left subtree**).



Deleting a Node

Deleting a node with two children (9).

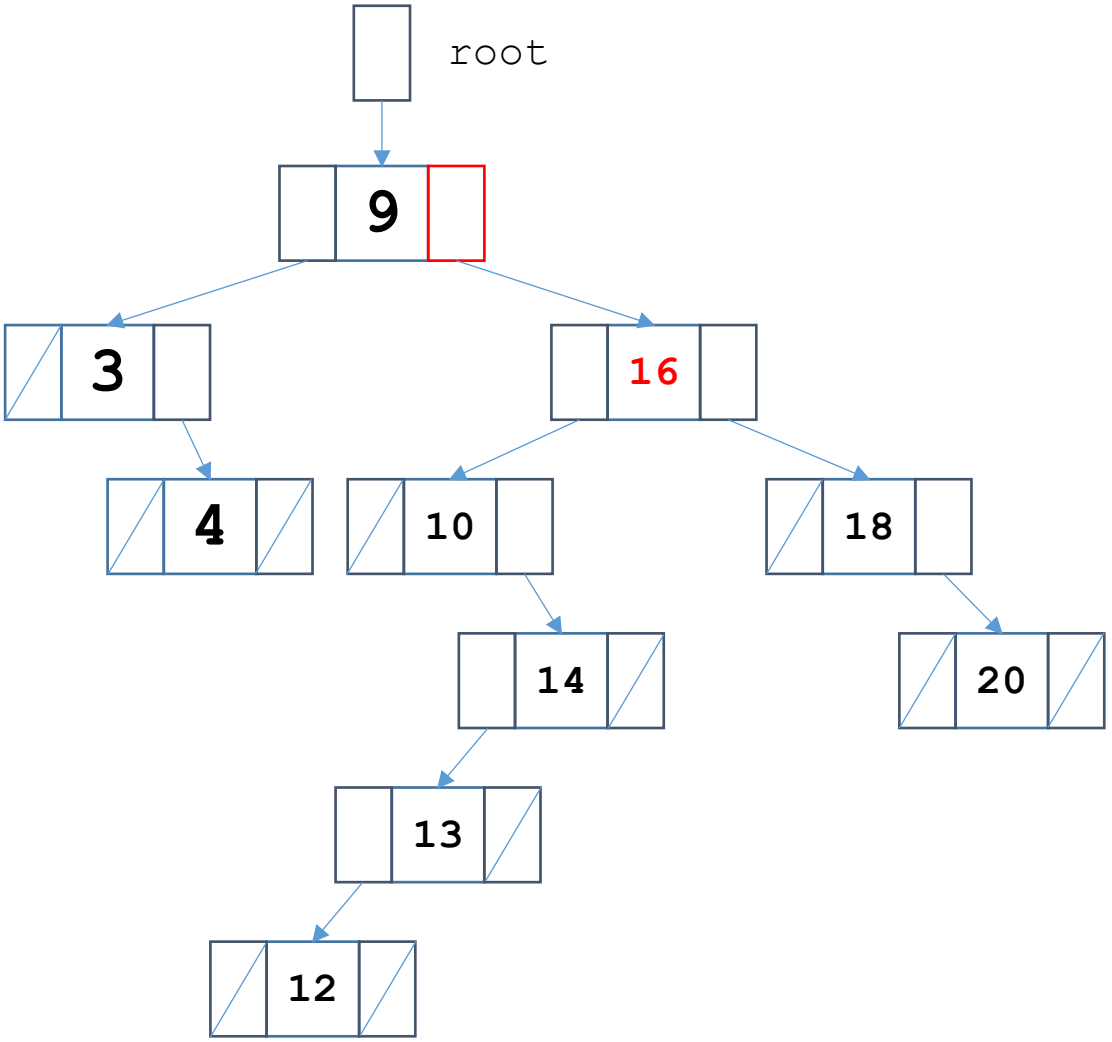
Replace it with its predecessor (rightmost nodes in the left subtree).



Deleting a Node

Deleting a node with two children (16).

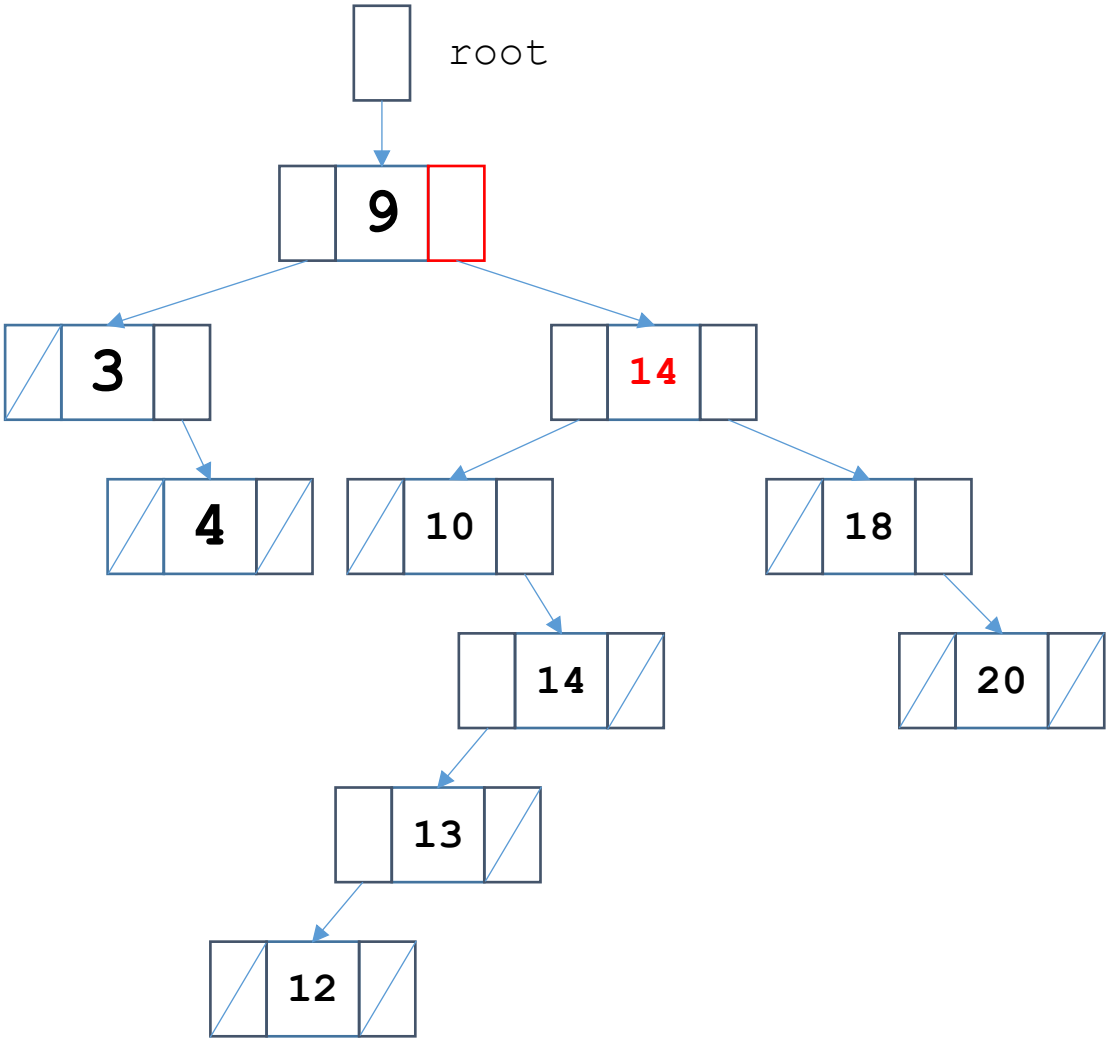
Replace it with its predecessor (rightmost nodes in the left subtree).



Deleting a Node

Deleting a node with two children (16).

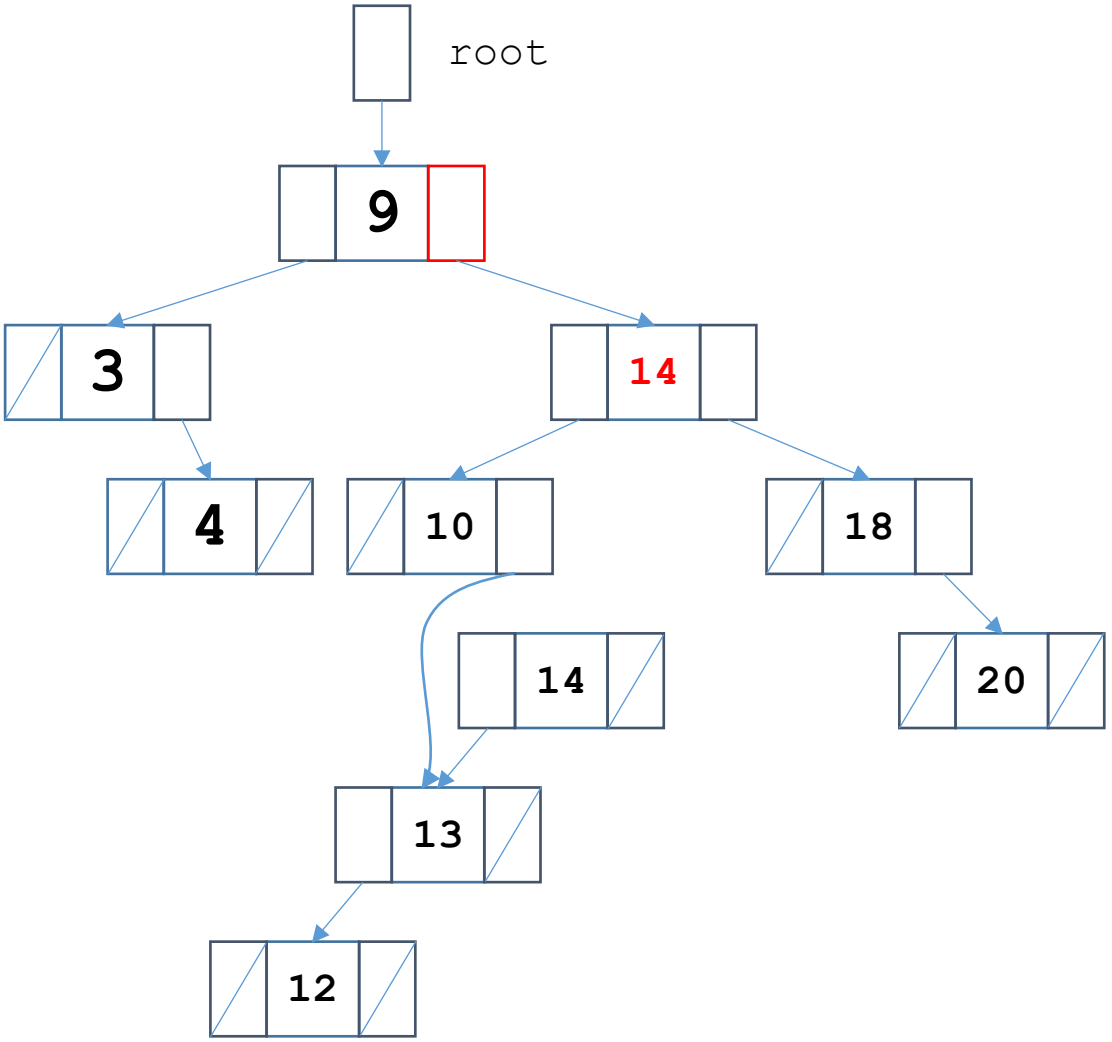
Replace it with its predecessor (rightmost nodes in the left subtree).



Deleting a Node

Deleting a node with two children (16).

Replace it with its predecessor (rightmost nodes in the left subtree).



Function DeleteItem

```
void DeleteNode(TreeNode*& tree);
// void subDelete(TreeNode*& tree, ItemType item);
void GetPredecessor(TreeNode* tree, ItemType& data);

void TreeType::DeleteItem(ItemType item)
{
    Delete(root, item);
}

void Delete(TreeNode*& tree, ItemType item)
{
   if (tree != NULL)
        {
              if (item < tree->info)
                     Delete(tree->left, item);
              else if (item > tree->info)
                      Delete(tree->right, item);
              else
                     DeleteNode(tree);
        }

}
```

Function DeleteItem

```
void DeleteNode(TreeNode*& tree)
{
    ItemType data;
    TreeNode* tempPtr;

    tempPtr = tree;
    if (tree->left == NULL && tree->right == NULL)
    {
        tree = NULL;
        delete tempPtr;
    }
    else if (tree->left == NULL)  // ensure that tree->right is NOT NULL
    {
        tree = tree->right;
        delete tempPtr;
    }
    else if (tree->right == NULL)
    {
        tree = tree->left;
        delete tempPtr;
    }
    else
    {
        GetPredecessor(tree->left, data);
        tree->info = data;
        Delete(tree->left, data);
    }
}
```

Function `DeleteItem`

```
void GetPredecessor(TreeNode* tree, ItemType& data)
{
    while (tree->right != NULL)
        tree = tree->right;
    data = tree->info;
}
```

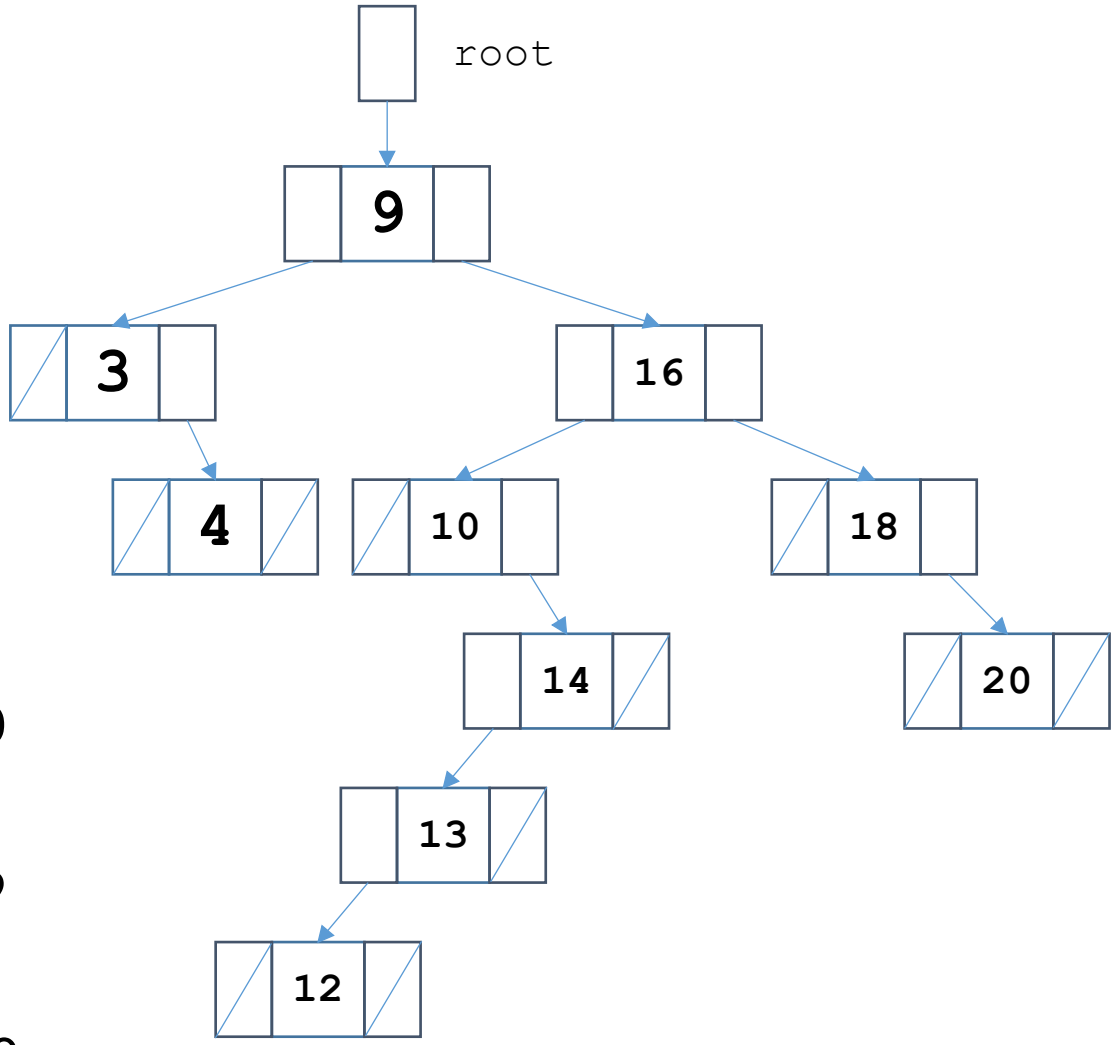
Traversal Techniques

- **Inorder traversal:** A systematic way of visiting all nodes in a binary tree that visits the nodes in the left subtree of a node, then visits the node, and then visits the nodes in the right subtree of the node

- **Postorder traversal:** A systematic way of visiting all nodes in a binary tree that visits the nodes in the left subtree of a node, then visits the nodes in the right subtree of the node, and then visits the node

- **Preorder traversal:** A systematic way of visiting all nodes in a binary tree that visits a node, then visits the nodes in the left subtree of the node, and then visits the nodes in the right subtree of the node

Traversal Techniques



- **Inorder**

  3 4 9 10 12 13 14 16 18 20
- **Postorder**

  4 3 12 13 14 10 20 18 16 9
- **Preorder**

  9 3 4 16 10 14 13 12 18 20

Thank You!!