CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

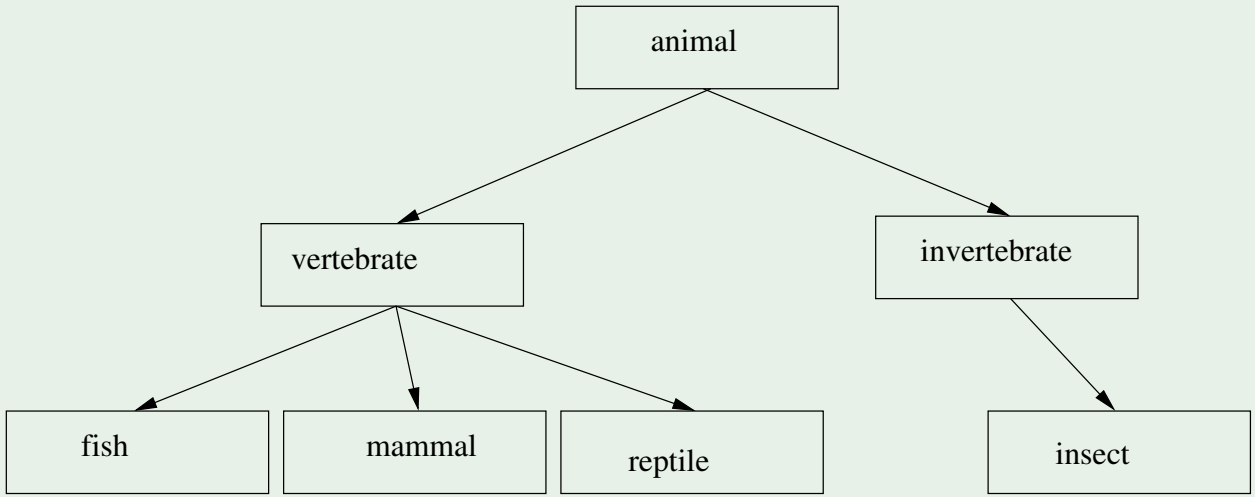
November 13, 2017

Outline

Outline

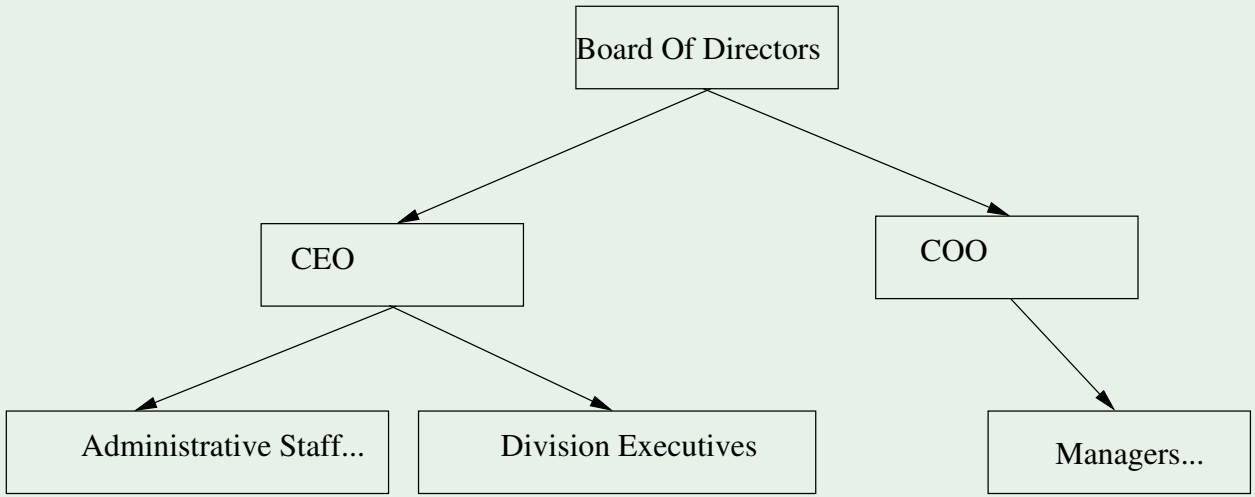
Uses Of Trees

Taxonomies



Uses Of Trees

Hierarchies



Uses Of Trees

Efficient Containers of Sequential Data

Each node represents a single data item of a collection organized, for efficient access, into a tree:

- Binary Search Trees
- Heaps
- Priority Queues
- B-Trees, Quad Trees, etc.

Definitions

Definitions I

- A tree consists of nodes connected by edges.
- A node can have zero or more child nodes. A child is connected to its parent node by a single edge.
- Children with the same parent are called siblings.
- A tree with no nodes or edges is an empty tree.
- A nonempty tree will have one special node with no parent called the root node.
- A node with no children is called a leaf.

Definitions

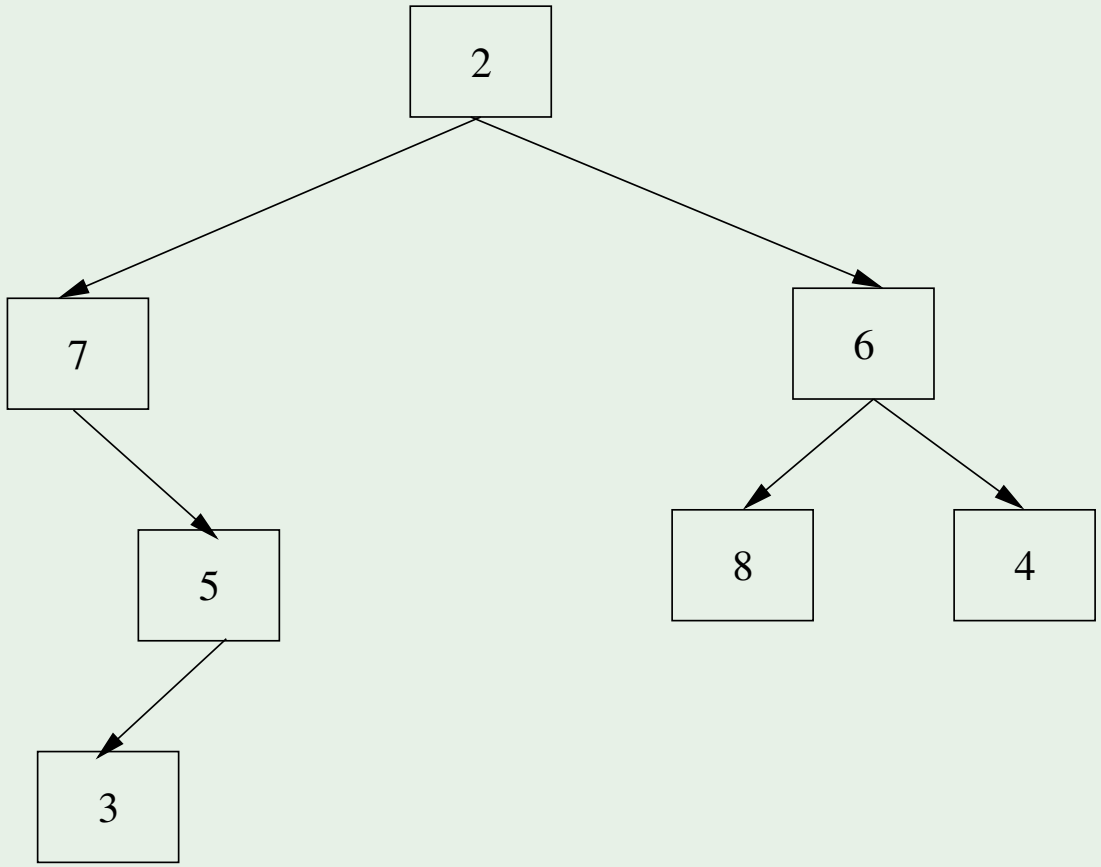
Definitions II

- All nodes are connected to the root by a path of edges.
- The depth of a node is the length of its path to the root. The root has depth zero.
- A level of a tree is a set of nodes which have the same depth.
- The descendants of a node are nodes whose paths include that node.
- The ancestors of a node are nodes on its path to the root.

Binary Trees

Binary Trees

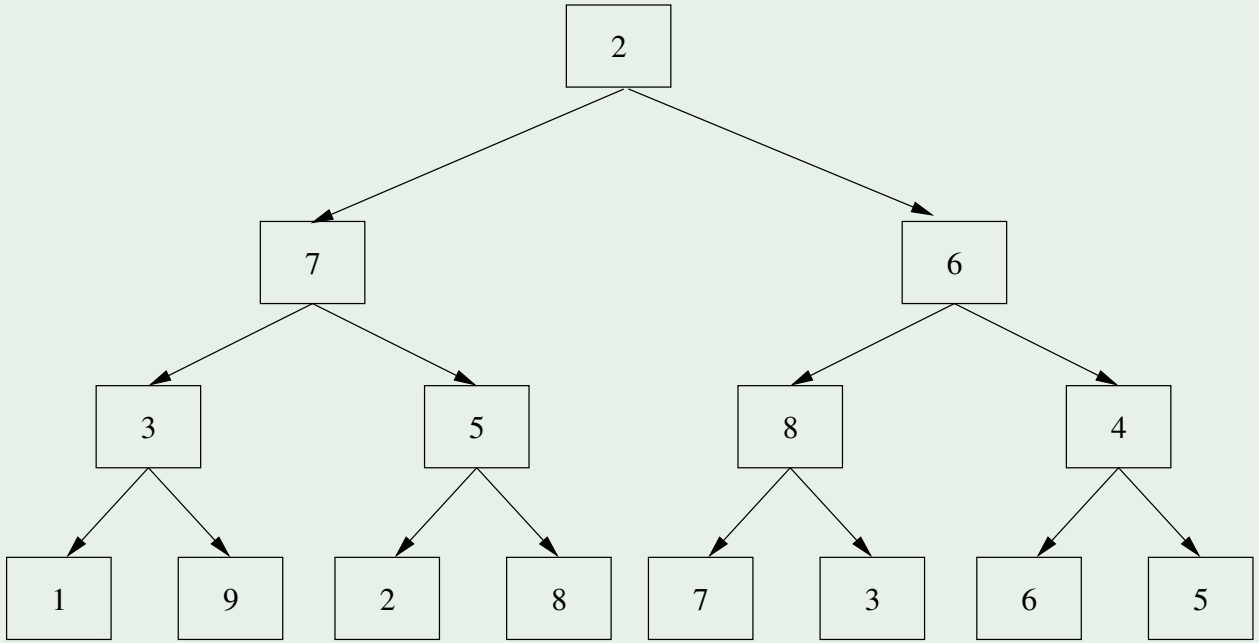
- A tree whose nodes have at most two children is a binary tree.



Binary Trees

Full Binary Trees

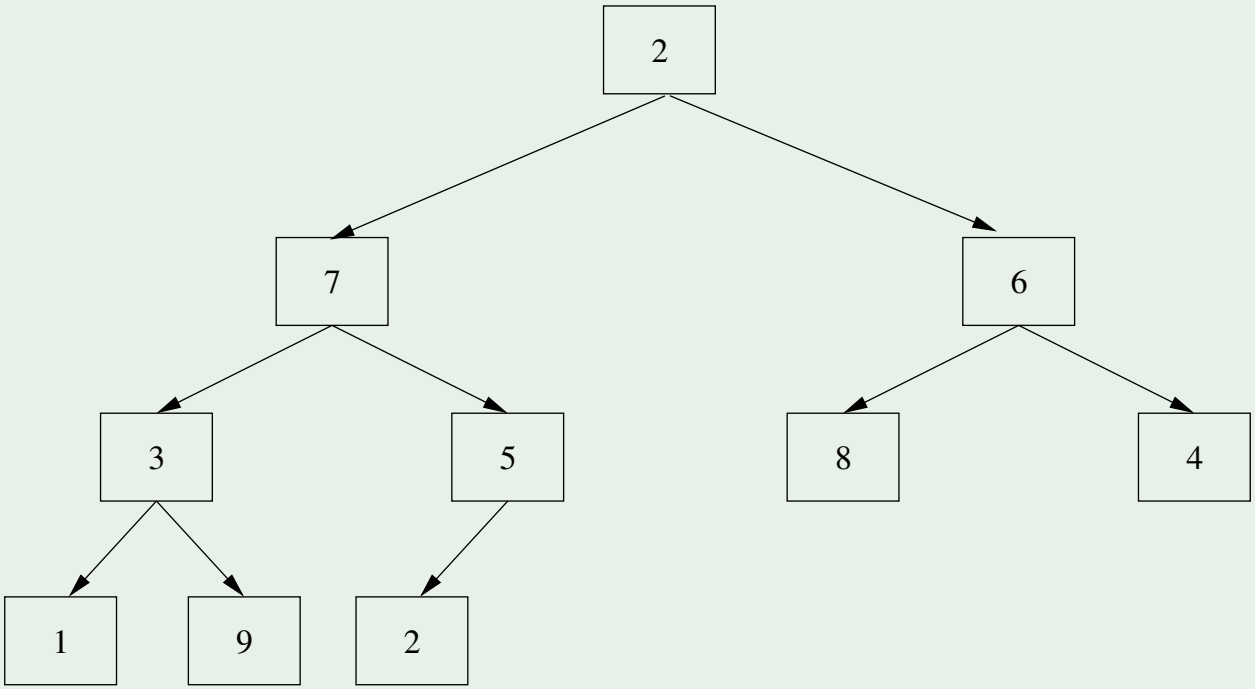
- A full binary tree is one whose levels have every position filled.



Binary Trees

Complete Binary Trees

- A complete binary tree has every level filled except the bottom, which is filled from left to right.



Binary Tree Traversal

Binary Trees are Recursive Data Structures

A binary tree can be defined as either

- An empty binary tree (base case!), or
- a node having two binary trees as attributes, a left subtree and a right subtree (recursive case).
- Using this recursive definition, recursive algorithms can be used to process the nodes of a binary tree.

Binary Tree Traversal

Preorder

```
def traverse(tree):
   if tree is not empty:
      process data at tree's root
      traverse(tree's left subtree)
      traverse(tree's right subtree)
```

Binary Tree Traversal

Inorder

```
def traverse(tree):
   if tree is not empty:
      traverse(tree's left subtree)
      process data at tree's root
      traverse(tree's right subtree)
```

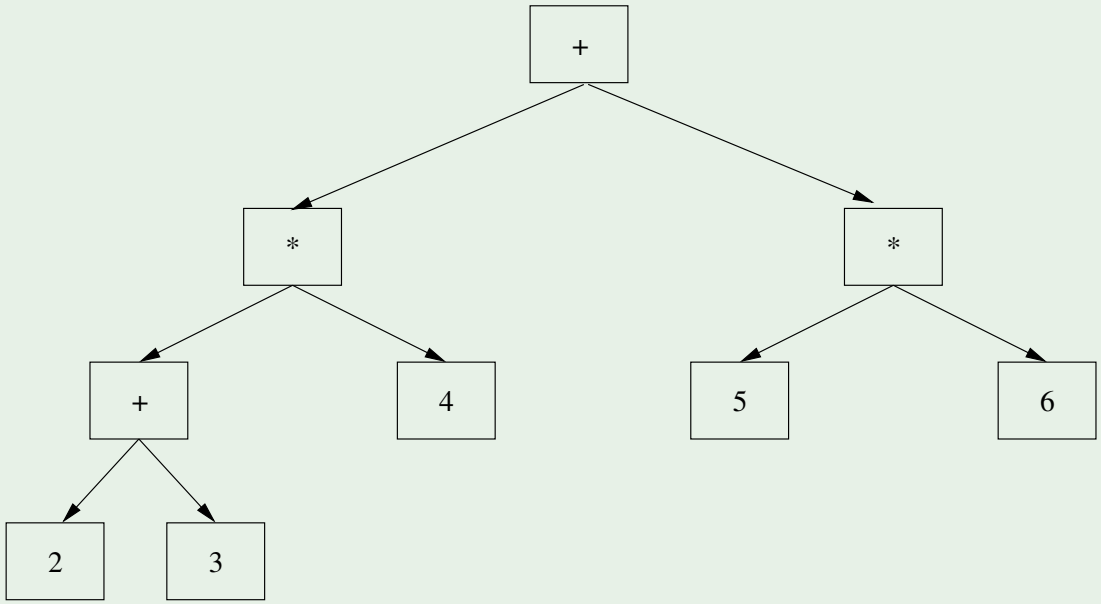
Binary Tree Traversal

Postorder

```
def traverse(tree):
   if tree is not empty:
      traverse(tree's left subtree)
      traverse(tree's right subtree)
      process data at tree's root
```

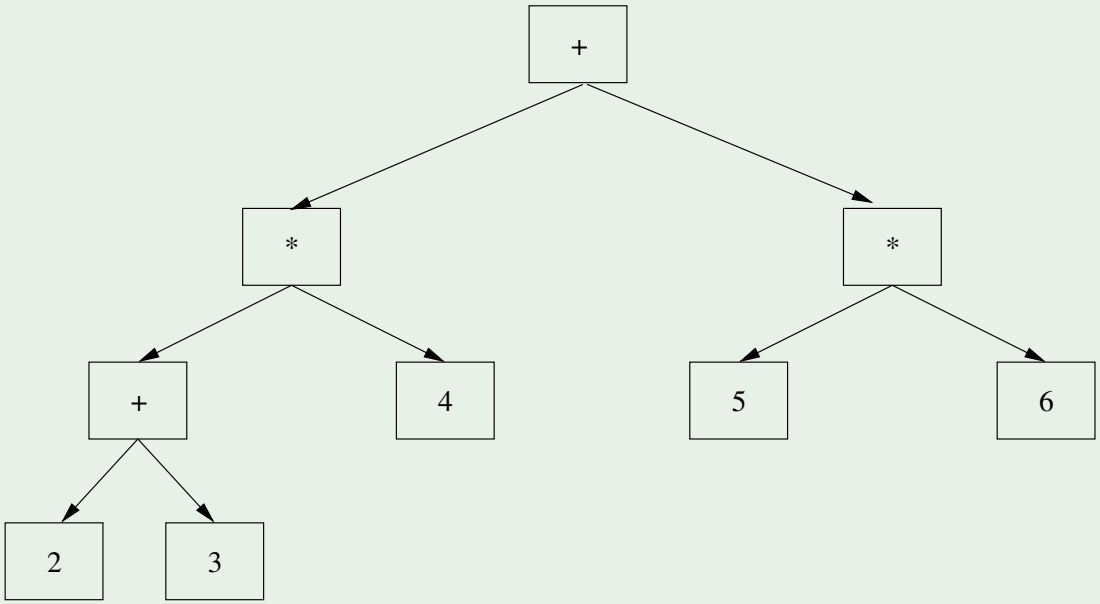
Binary Tree Representation Of An Expression

Abstract Syntax Tree



Binary Tree Representation Of An Expression

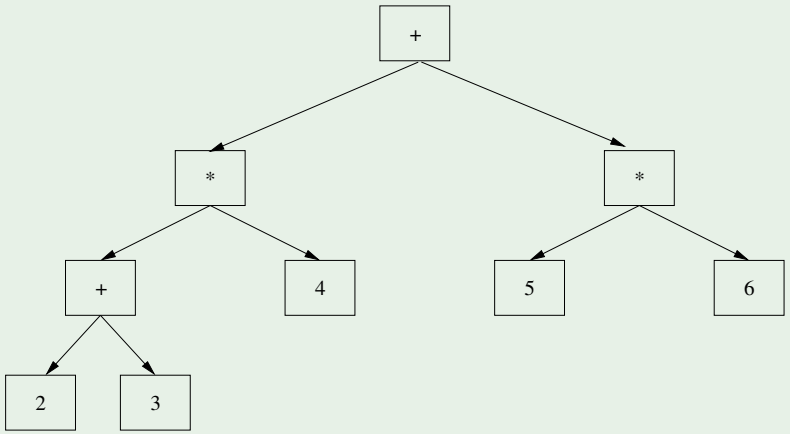
Different Traversals Give The Different Notations



- Printing in preorder gives Prefix Notation.
- Printing inorder gives Infix Notation.
- Printing in Postorder gives Postfix Notation.

Binary Tree Representation Of An Expression

Evaluation




```
def evaluateTree(tree):
  if tree's root is an operand:
     return root data
  else:  # root contains an operator
     leftValue = evaluateTree(tree's left subtree)
     rightValue = evaluateTree(tree's right subtree)
     result = operator(leftValue, rightValue)
     return result
```

Linked Representation Of A Binary Tree in C++

Use A TreeNode Class

- A binary tree has a single instance variable, `_root`, a pointer to a `TreeNode`.
- A nonempty binary tree is represented with `_root` as a pointer to an actual `TreeNode` object.
- An empty binary tree has no root. We represent this by using a value of NULL for the instance variable `_root`. as a pointer to a `TreeNode`.
- The TreeNode class is defined recursively.

Linked Representation Of A Binary Tree in C++

The TreeNode class is defined recursively.

- The `_left` and `_right` instance variables are pointers to `TreeNode` objects, if there are left or right subtrees, or `NULL` if those subtrees are empty.
- The `_item` instance variable is the data item stored in the TreeNode.

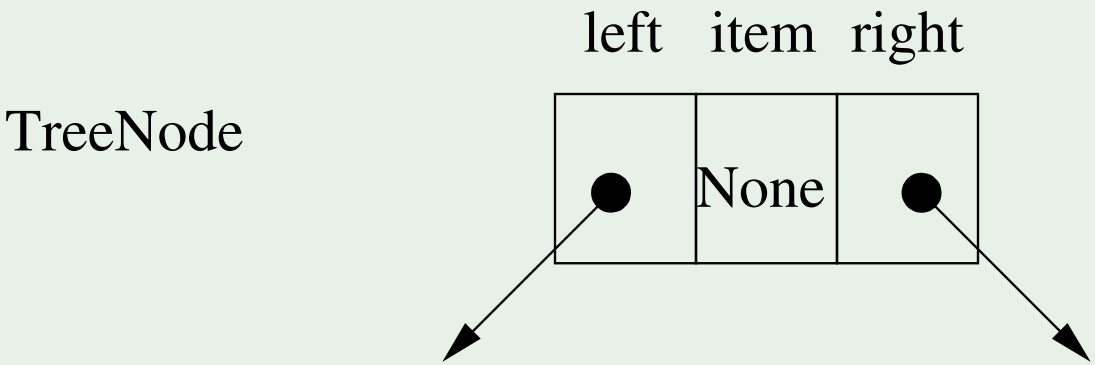
Linked Representation Of A Binary Tree in C++

(Simplified) Definition Of A TreeNode Class in C++

```
class TreeNode {
public:
   TreeNode(int item, TreeNode* left = NULL,
TreeNode* right = NULL);
   ~TreeNode(); // destructor
   int _item; // value
   TreeNode* _left; // link to left subtree
   TreeNode* _right; // link to right subtree
};
```

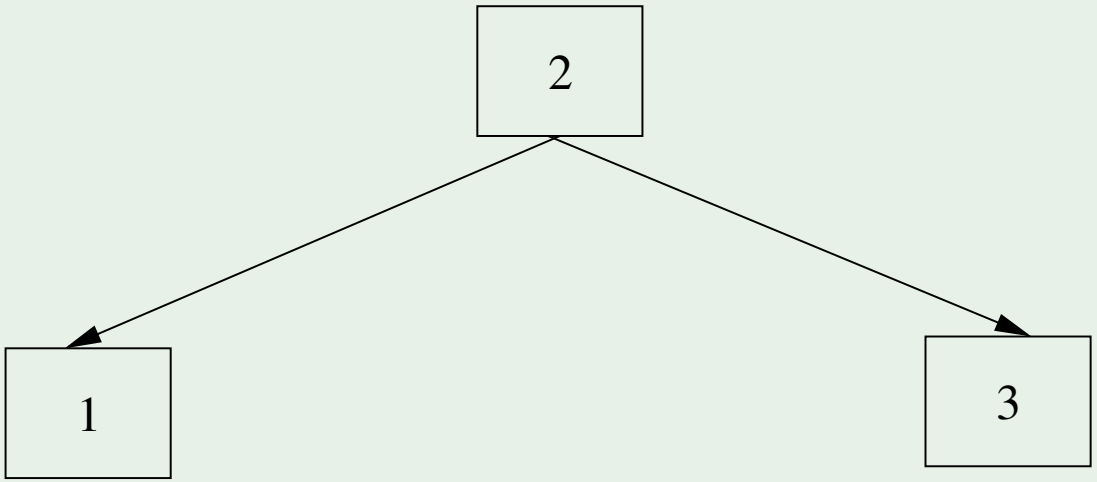
Linked Representation Of A Binary Tree in C++

C++ Examples Using The TreeNode Class



Linked Representation Of A Binary Tree in C++

Example 1: Abstract View



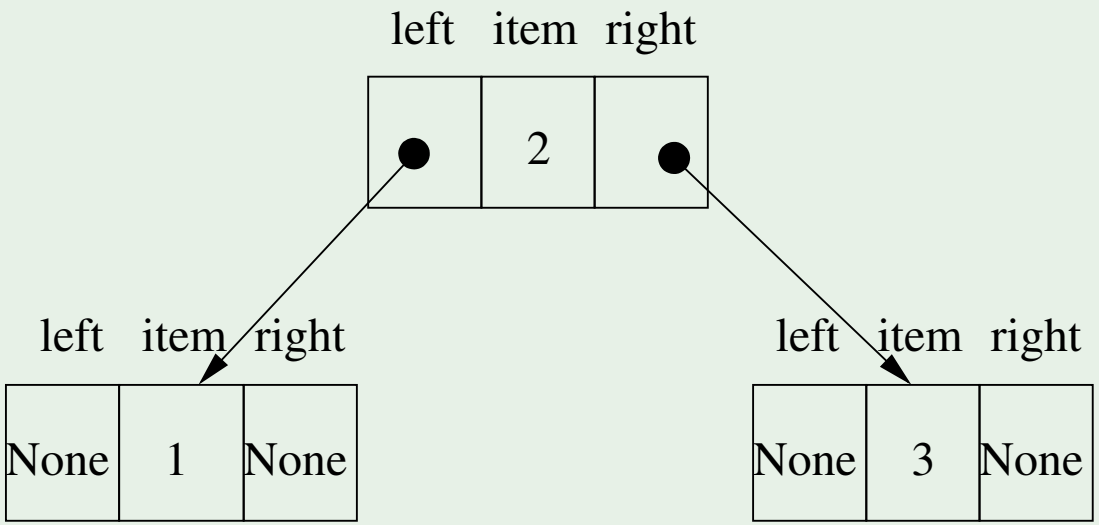
Linked Representation Of A Binary Tree in C++

Example 1: C++ Implementation

```
TreeNode *_left = new TreeNode(1);
TreeNode *_right = new TreeNode(3);
TreeNode *_root = new TreeNode(2, _left, _right);
```

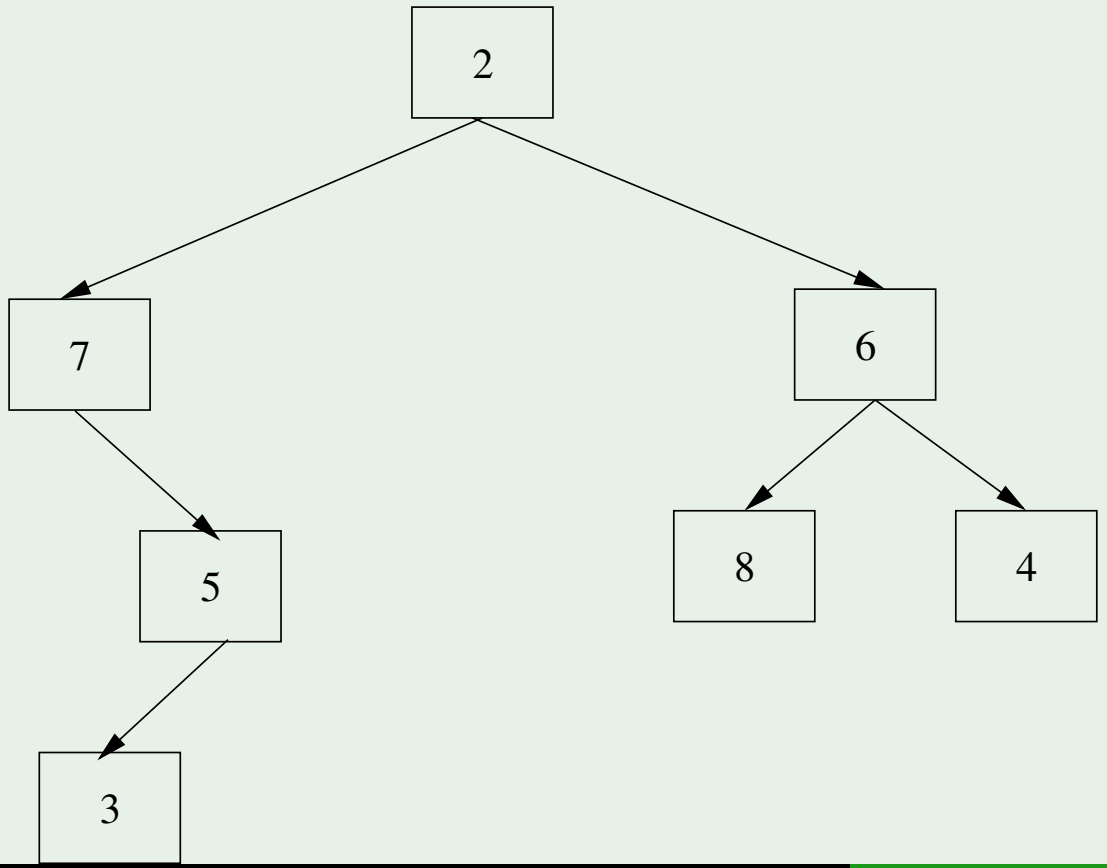
Linked Representation Of A Binary Tree in C++

Example 1: C++ Memory Model



Linked Representation Of A Binary Tree in C++

Example 2: Abstract View



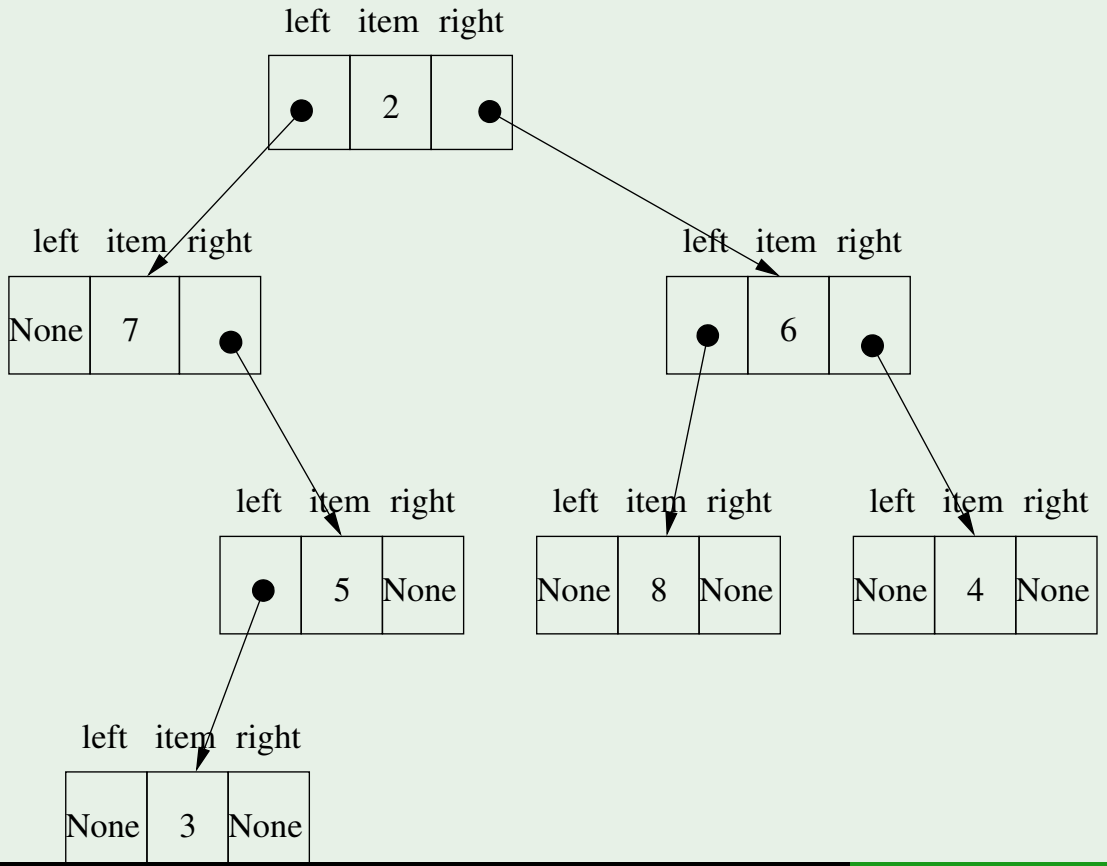
Linked Representation Of A Binary Tree in C++

Example 2: C++ Implementation

```
_root = new TreeNode(2,
          new TreeNode(7,
             NULL,
             new TreeNode(5, new TreeNode(3))),
          new TreeNode(6, new TreeNode(8), new
TreeNode(4)));
```

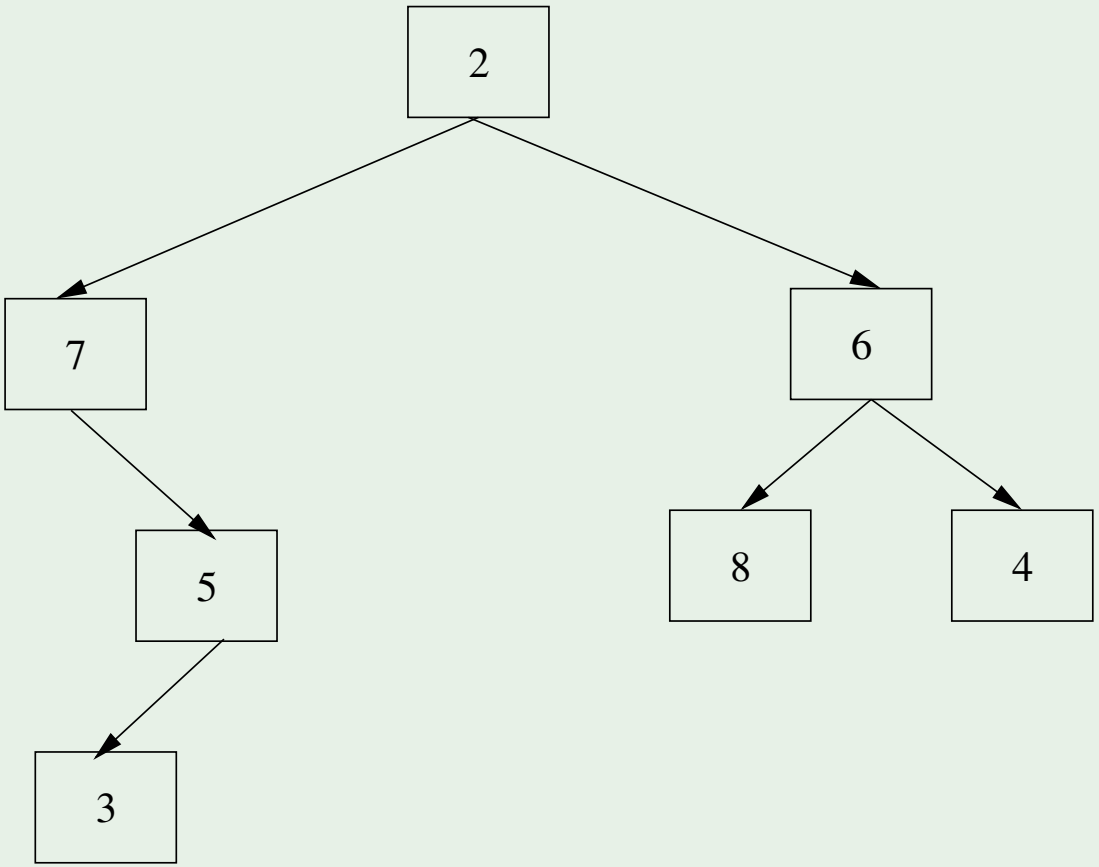
Linked Representation Of A Binary Tree in C++

Example 2: C++ Memory Model



Array Representation of Binary Trees

Abstract View



Array Representation of Binary Trees

Implementation

2	7	6	NULL	5	8	4	NULL	NULL	3
1	2	3	4	5	6	7	8	9	10

```
def left_child(i):
   return 2 * i
def right_child(i):
   return 2 * i + 1
def parent(i):
   return i // 2
```