Trees
Why should you care?

Trees are used to organize data in many software applications, including:

Databases (B-TREES)
Data Compression (Huffman Trees)
Bitcoin (Merkle Trees)
Medical Diagnosis (Decision Trees)

And because you'll be asked about them in job interviews and on exams.

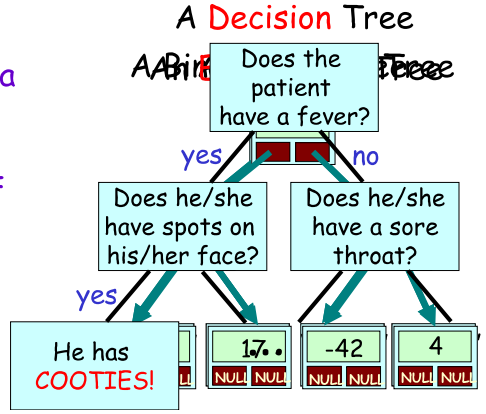
So pay attention!

Trees

"I think that I shall never see a data structure as lovely as a tree." - Carey Nachenberg

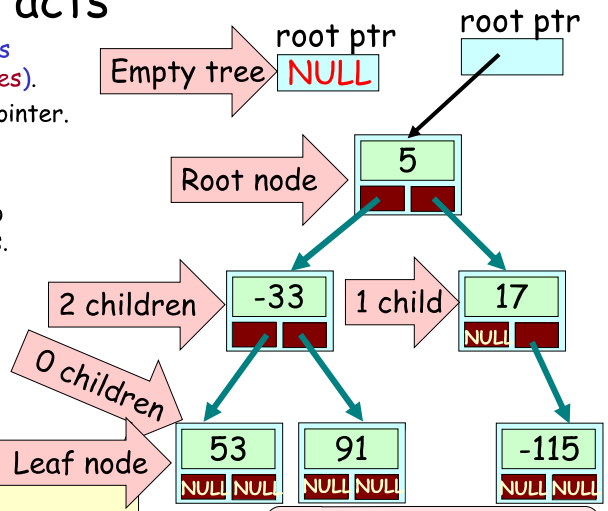
A Tree is a special linked list-based data structure that has many uses in Computer Science:

- To organize hierarchical data
- To make information easily searchable
- To simplify the evaluation of mathematical expressions
- To make decisions



Basic Tree Facts

1. Trees are made of nodes (just like linked list nodes).
2. Every tree has a "root" pointer.
3. The top node of a tree is called its "root" node.
4. Every node may have zero or more "children" nodes.
5. A node with 0 children is called a "leaf" node.
6. A tree with no nodes is called an "empty tree."




```
struct node
{
  int value;  // some value

  node *left, *right;
};

node *rootPtr;
```

But instead of just one next pointer, a tree node can have two or more next pointers!

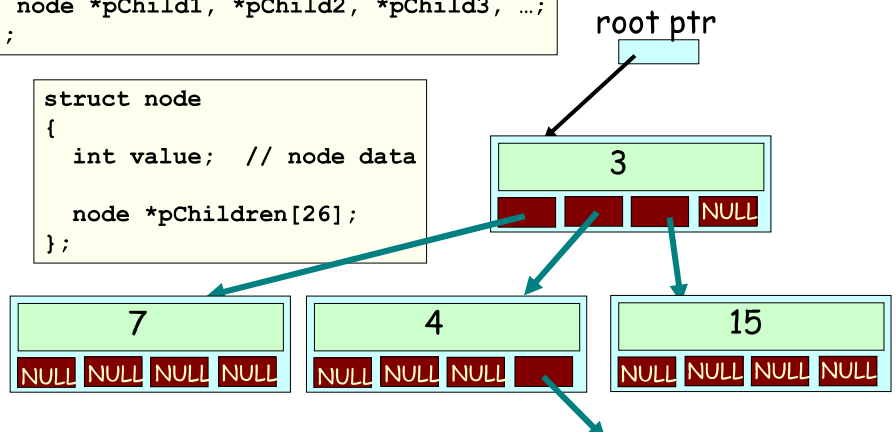
The tree's root pointer is like a linked list's head pointer!

Tree Nodes Can Have Many Children

A tree node can have more than just two children:

```
struct node
{
  int value;  // node data

  node *pChild1, *pChild2, *pChild3, …;
};
```

```
struct node
{
  int value;  // node data

  node *pChildren[26];
};
```



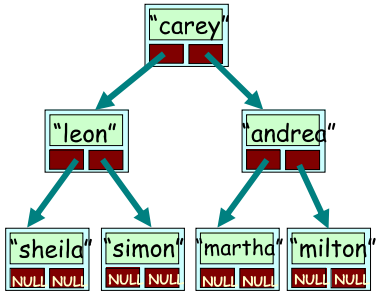
Binary Trees

A binary tree is a special form of tree. In a binary tree, every node has at most two children nodes:

A left child and a right child.

```
struct BTNODE  // binary tree node
{
  string value;  // node data

  BTNODE *pLeft, *pRight;
};
```

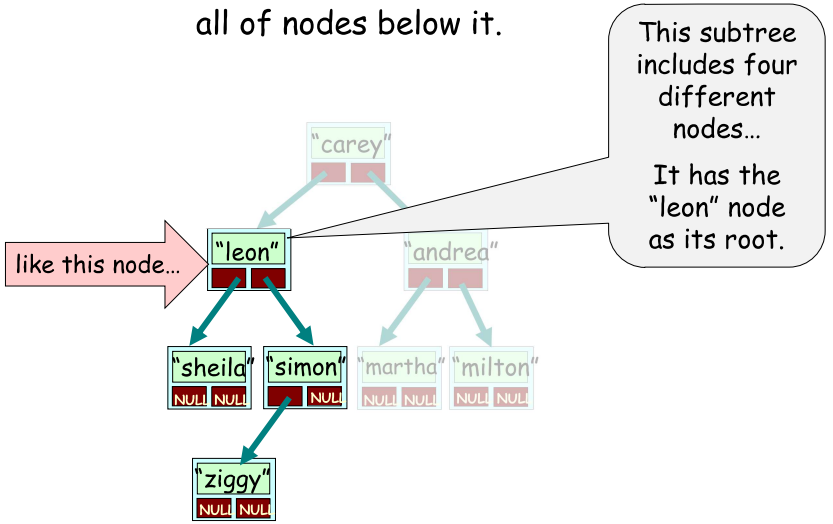
A Binary Tree



Binary Tree Subtrees

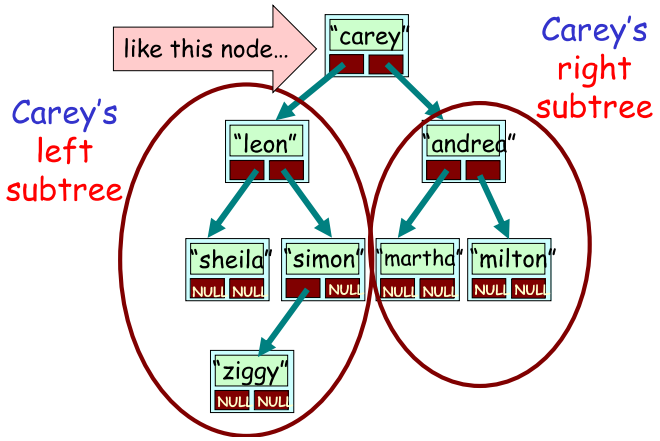
We can pick any node in the tree…

And then focus on its "subtree" - which includes it and all of nodes below it.



Binary Tree Subtrees

If we pick a node from our tree…
we can also identify *its* left and right sub-trees.



Operations on Binary Trees

The following are common operations that we might perform on a Binary Tree:

- enumerating all the items
- searching for an item
- adding a new item at a certain position on the tree
- deleting an item
- deleting the entire tree (destruction)
- removing a whole section of a tree (called **pruning**)
- adding a whole section to a tree (called **grafting**)

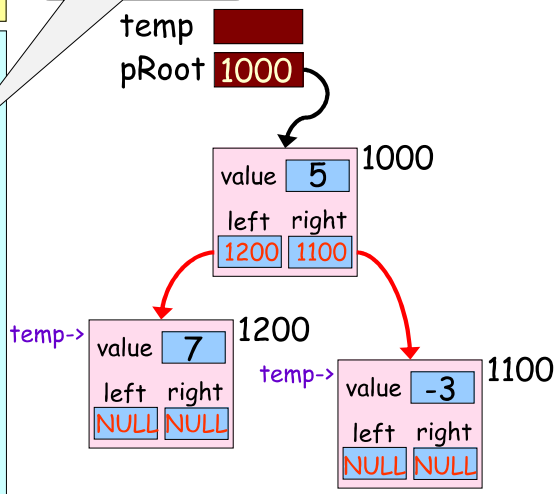
We'll learn about many of these operations over the next two classes.

```
struct BTNODE  // node
{
  int value;  // data
  BTNODE *left, *right;
};
```

```
main()
{
   BTNODE *temp, *pRoot

   pRoot = new BTNODE;
   pRoot->value = 5;

   temp = new BTNODE;
   temp->value = 7;
   temp->left = NULL;
   temp->right = NULL;
   pRoot->left = temp;

   temp = new BTNODE;
   temp->value = -3;
   temp->left = NULL;
   temp->right = NULL;
   pRoot->right = temp;
   // etc…
```

As with linked lists, we use dynamic memory to allocate our nodes.

A Simple Tree Example



And of course, later we'd have to delete our tree's nodes.

We've created a binary tree… now what?

Now that we've created a binary tree, what can we do with it?

Well, next class we'll learn how to use the binary tree to speed up searching for data.

But for now, let's learn how to iterate through each item in a tree, one at a time.

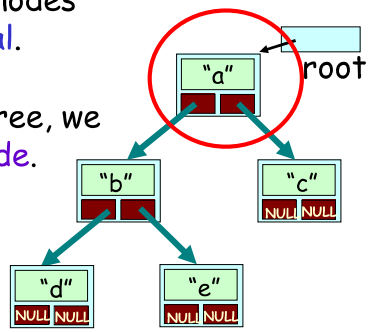
This is called "traversing" the tree, and there are several ways to do it.

Binary Tree Traversals

When we iterate through all the nodes in a tree, it's called a traversal.

Any time we traverse through a tree, we always start with the root node.

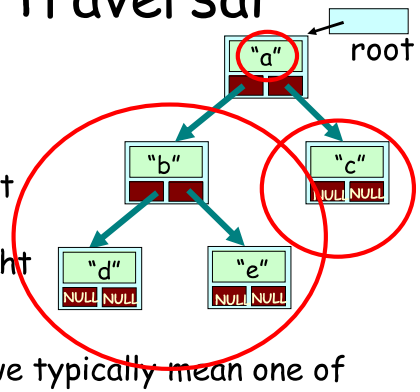
There are four common ways to traverse a tree.



Each technique differs in the order that each node is visited during the traversal:

1. Pre-order traversal
2. In-order traversal
3. Post-order traversal
4. Level-order traversal

The Preorder Traversal



Preorder:

1. Process the current node.
2. Process the nodes in the left sub-tree.
3. Process the nodes in the right sub-tree.

By "process the current node" we typically mean one of the following:

1. Print the current node's value out.
2. Search the current node to see if its value matches the one you're searching for.
3. Add the current node's value to a total for the tree
4. Etc…