

Binary Trees

To implement binary trees, we have two prototypes:

`BTNode` and `BinaryTree`.

A `BTNode` is very much like a `ListNode` except that, instead of having one instance variable whose value is a reference, it has two: `leftChild` and `rightChild`.

We define `get` and `set` methods for all three instance variables.

If the constructor is called with fewer than three arguments, `rightChild` is set to `null`.

If the constructor is called with fewer than two arguments, `leftChild` is also set to `null`.

The `toString` method simply returns what's returned by the `toString` method of whatever is the value of the data instance variable.

The `inorderTraversal` property of the prototype is set to the `BTNode_inorderTraversal` function defined outside the constructor.

This method performs an inorder traversal of the binary tree rooted at the current `BTNode`.

As long as the left child is not `null`, it recursively invokes the `inorderTraversal` method of the left child.

Then it outputs the value of the `data` instance variable.

Finally, as long as the right child is not `null`, it recursively invokes the `inorderTraversal` method of the right child.

```
function BTreeNode( data, leftChild, rightChild )
{
    this.data = data;
    this.leftChild = leftChild || null;
    this.rightChild = rightChild || null;
    this.getdata =
        function() { return this.data; };
    this.setdata =
        function( data ) { this.data = data; };
    this.getleftChild =
        function() { return this.leftChild; };
    this.setleftChild =
        function( leftChild )
        { this.leftChild = leftChild; };
    this.getrightChild =
        function() { return this.rightChild; };
    this.setrightChild =
        function( rightChild )
        { this.rightChild = rightChild; };
    this.toString =
        function() { return this.data.toString(); };
    this.inorderTraversal = BTreeNode_inorderTraversal;
}

function BTreeNode_inorderTraversal()
{
    if ( this.leftChild != null )
        this.leftChild.inorderTraversal();

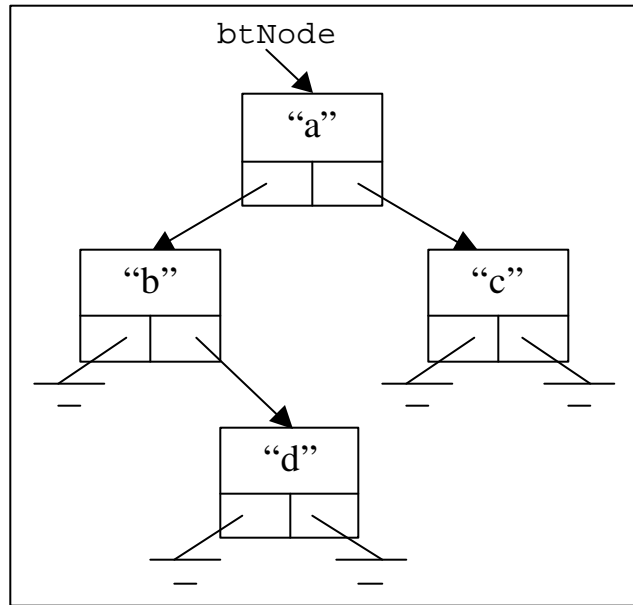
    document.writeln( this.data + "<br>" );

    if ( this.rightChild != null )
        this.rightChild.inorderTraversal();
}
```

The following is the file that tests BTNode.

It creates the binary tree shown at right, with btNode a reference to its root.

It then invokes the `inorderTraversal` method of `btNode`.



```

var btNode = new BTNode( "a",
                        new BTNode( "b",
                                    null,
                                    new BTNode( "d" ) ),
                        new BTNode( "c" ) );

btNode.inorderTraversal();

```

The following is the HTML file that executes these two files.

`btNode.js` contains the `BTNode` constructor.

`TestBTNode.js` contains the test code.

```

<html>
<head>
<title>Tests</title>
<script type = "text/javascript"
        src = "btNode.js">
</script>
<script type = "text/javascript"
        src = "testBTNode.js">
</script>
</head>
</html>

```

The following is the rendering:

b
d
a
c

A `BinaryTree` has one instance variable, `root`, which is a reference to the root node of a binary tree made up of `BTNode` nodes.

Instances of this prototype are basically just packages for binary trees made up of `BTNode` nodes.

The constructor has one formal parameter, `rootData`.

If the constructor is invoked without any argument, `rootData` is undefined, and the root instance variable is set to `null`.

If a value is supplied for `rootData`, then this value is passed to the `BTNode` constructor, and the new `BTNode` is assigned to the `root` instance variable.

The code to do this is

```
this.root =
    rootData ? new BTNode( rootData ) : null;
```

We define `get` and `set` methods for the root.

An `inorder` traversal method is defined for binary trees.

This calls the `inorderTraversal` method of `root` as long as `root` isn't `null`.

It encloses the output generated by this method of `root` in `<p>` tags.

Conceptually, a binary tree is a recursive structure: its children are binary trees.

We should thus have a method that constructs a binary tree from left and right subtrees and data for the root.

We define a class method for this that has three formal parameters:

`rootData`: the data value for the new root

`leftSubT`, `rightSubT`: references to the left and right subtrees

The first step is to create a new `BinaryTree` with `rootData` as the value of the `data` instance variable of the root node:

```
bt = new BinaryTree( rootData );
```

Next, we have to “unpackage” the reference to the root of the left subtree:

```
leftSubT.root
```

This is assigned to the `leftChild` instance variable of the root node of our new `BinaryTree` object:

```
bt.root.leftChild = leftSubT.root;
```

A similar statement sets the `rightChild` instance variable of the root to a reference to the root of the right subtree:

```
bt.root.rightChild = rightSubT.root;
```

The entire file is shown on the next page.

```
function BinaryTree( rootData )
{
  this.root =
    rootData ? new BTNode( rootData ) : null;
  this.setroot =
    function( root ) { this.root = root; };
  this.getroot =
    function() { return this.root; };
}

function BinaryTree_inorderTraversal()
{
  document.writeln( "<p>" );

  if ( this.root != null )
    this.root.inorderTraversal();

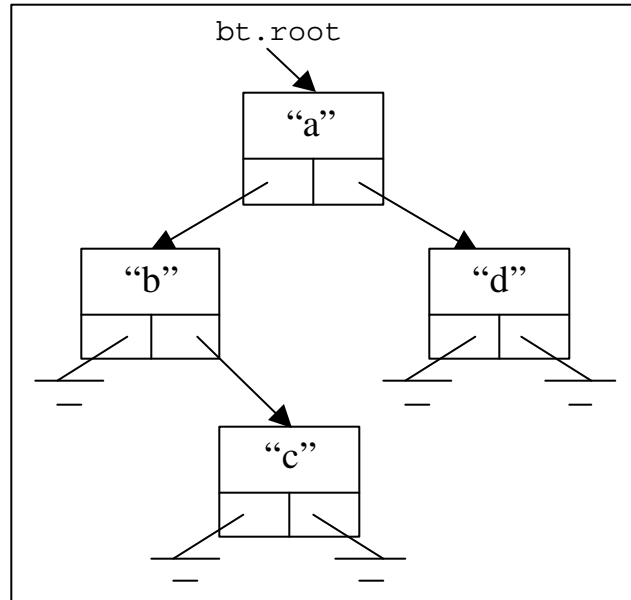
  document.writeln( "</p>" );
}
BinaryTree.prototype.inorderTraversal =
  BinaryTree_inorderTraversal;

function BinaryTree_makeBT( rootData, leftSubT,
                           rightSubT )
{
  bt = new BinaryTree( rootData );
  bt.root.leftChild = leftSubT.root;
  bt.root.rightChild = rightSubT.root;
  return bt;
}
BinaryTree.makeBT = BinaryTree_makeBT;
```

The following is test code for BinaryTree.

It creates the binary tree shown at right and assigns to bt the BinaryTree that packages a reference to the root of this tree.

It then invokes the inorderTraversal method of bt.



The constructor BinaryTree is used to construct singleton binary trees.

BinaryTree.makeBT is used to construct a binary tree with at least one non-null subtree.

```

var bt = BinaryTree.makeBT( "a",
    BinaryTree.makeBT( "b",
        new BinaryTree( null ),
        new BinaryTree( "c" ) ),
    new BinaryTree( "d" ) );

bt.inorderTraversal();

```

The following is the HTML file that executes both these files and the file with the definition of `BTNode` (since `BinaryTree` uses `BTNode`).

`btNode.js` contains the `BTNode` constructor.

`binaryTree.js` contains the `BinaryTree` constructor.

`TestBinaryTree.js` contains the test code.

```
<html>
<head>
<title>Tests</title>
<script type = "text/javascript"
        src = "btNode.js">
</script>
<script type = "text/javascript"
        src = "binaryTree.js">
</script>
<script type = "text/javascript"
        src = "testBinaryTree.js">
</script>
</head>
</html>
```

The following is the rendering:

```
b
c
a
d
```