CS 180 Problem Solving and OO Programming

Fall 2011

Recitation Week 13: December 1-2, 2011
Recursion and dynamic data structures

Problem 1: The Fibonnaci sequence is defined as follows:

Fib(0)=0;
Fib(1)=1;
Fib(n)=Fib(n-1)+Fib(n-2);

Write a recursive method in Java named Fib that takes integer **n>=0** as input and returns the n^{th} Fibonnaci number.

Problem 2: Let GCD(x, y) denote the greatest common divisor of two integers x>=0 and **y>=0**. For example GCD(15, 12)=3, an GCD(15, 7)=1. A recursive definition of GCD is as follows.

GCD(x, y) = x, if y=0;

= GCD(y, x%y); otherwise

Write a Java method named GCD that returns the GCD of two integers x>=0 and y>=0.

Problem 3:

A binary search tree is a binary tree such that (a) there is a unique node known as the *root* of the tree, (b) each node has at most two *descendants* one known as the *left* descendent and the other as the *right* descendant, and (c) for any two nodes n1 and n2 in the tree if the data contained in n1 is less than the data in node n2 then n1 must appear to the left of n1; if the data contained in n1 is greater than the data in node n2 then n1 must appear to the right of n2, and (d) no two nodes in the tree have the same data value.

(a) Write a class named `Node`. It has three private attributes: `val` (an integer), `leftLink`, and `rightLink`. A `Node` object with value `v` and both links set to `null` can be created as follows:

```
new Node (v, null, null);
```

Add the get and set methods to the `Node` class as follows.

getLeftLink() returns `leftLink`

getRightLInk() returns `rightLink`

`setLeftLink(Node l)` sets `leftLink to l`

`setRightLink(Node r)` sets `rightLink to r.`

(b) Write a class named `BinarySearchTree`. This class has one private attribute named `root` of type `Node`. An empty binary tree can be constructed as follows:

`new BinarySearchTree`();
This sets `root` to null. Add the following methods to the `BinarySearchTree` class.

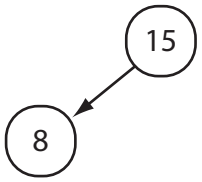
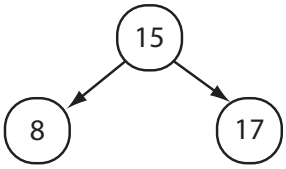
`public void` addNode(`Node` v): This adds a new node with value `v` to the tree.
`public void` addNode(Node n, Node r): This adds a new node n to the tree rooted at node `r`.

`public void` traverse(): If the tree is empty, this method simply returns else it calls the other traverse() method with root as the input that recursively traverses the tree.

`public void` traverse(Node r): This traverses the tree in in-order starting at root r and prints the nodes. The output will be a sorted list of integers in ascending order.

`public void` traverse(Node r): This traverses the tree in in-order. And prints the nodes. The output will be a sorted list of integers in ascending order.

The following example shows the effect of calling the various methods to construct the tree.

Method call	Consequence	Tree
`BinarySearchTree` t= `new BinarySearchTree` ();	A new `BinarySearchTree` object is created.	Empty, root is set to null.
t.addNode(`new Node`(15, null, null))	A new node with value 15 is added to the tree. Root is set to this node.	Root→ 
t.addNode(`new Node`(8, null, null))	Another node with value 8 is added to the tree. This is done by calling `addNode`() method with the new node and the root (node with 15) as inputs	
t.addNode(`new Node`(17, null, null))	Another node with value 17 is added to the tree. This is done by calling `addNode`() method with the new node and the root as inputs	

t.traverse()	Tree traversed and data printed	8 15 17

Note that the addNode() method is overloaded. The method with one parameter is called to add a node to the tree. If the tree is empty it simply adds a new node to the tree and makes it the root. Otherwise it calls the other addNode() method with the node to be added as well as the root as input. This second addNode() method then recursively traverses the tree to find the node where the new node is to be added.

<End of Problems for Week 15>