Computer Science II — CSci 1200
Lab 13 — EXTRA CREDIT
Binary Search Trees and Recursion

Introduction

This lab explores binary search trees and the use of recursive functions to manipulate them. Once you have a basic understanding of trees, the actual code that you have to write for the lab is quite short. Review the Lecture 23 notes prior to starting.

Download the files:

`http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab13/tree.h`
`http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab13/main.cpp`

Then, turn off all network connections.

Examine the code in `tree.h` and `main.cpp`. The tree manipulation functions are not hidden inside a class, as they are in the standard library. Instead, the functions are called with a pointer to the root of the tree as an argument. When the function might have to change the root, as insert sometimes does, this root pointer must be passed by reference. Class implementations, such as the set and map containers, maintain a pointer to the root of the tree internally. Member functions, similar to the ones you will implement in this lab, that traverse or manipulate the tree are implemented as private functions and are passed a pointer to the root by a public member function.

Only a few functions are provided in the code, but they are enough to get started and do the lab. In particular, an insert function is provided to place values in the tree in such a way as to maintain the binary search tree ordering. A simple print function is provided that prints a tree sideways. If you examine the sequence of calls to insert from the main program and then examine the output from the main program you can begin to see how it works. Spend some time playing with this, changing the sequence of insert operation in the main function and seeing the resulting tree structure.

Now you are ready to work through the checkpoints.

Checkpoints

1. Write two recursive functions. The first, `find`, returns true if and only if a value is stored in the binary search tree. The function must exploit the ordering property to make the search efficient. Its prototype is

```
template <class T>
bool find( TreeNode<T> * root, T const& value );
```

The second adds the values stored in the tree. Its prototype is

```
template <class T>
T add( TreeNode<T> * root );
```

Add code to the main function to test these two functions.

2. Write and test a non-recursive version of `find`. It should have the same prototype as the recursive version.

3. Write a recursive function that copies a tree, returning a pointer to the root of the new tree. Its prototype should be

```
template <class T>
TreeNode<T>* copy( TreeNode<T> * root );
```

Think of this problem in terms of a pre-order traversal of the tree.