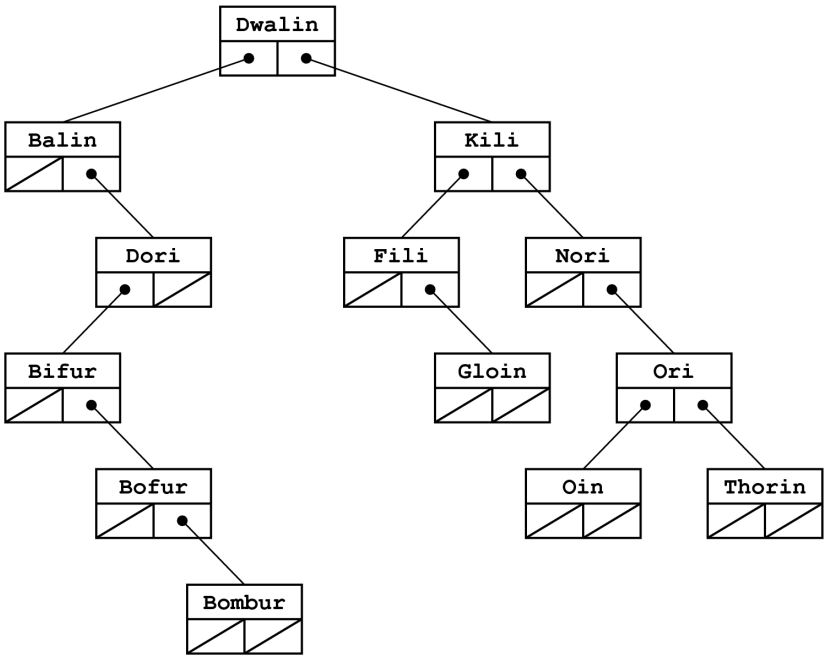
Eric Roberts
CS 106B

Handout #38A
February 18, 2015

Solutions to Section Handout #6

1. Tracing binary tree insertion



1a. What is the height of the resulting tree? **5**

1b. Which nodes are leaves? **`Bombur`**, **`Gloin`**, **`Oin`**, and **`Thorin`**

1c. Which nodes are out of balance? **`Balin`**, **`Bifur`**, **`Dori`**, **`Dwalin`**, **`Kili`**, and **`Nori`**

1d. Which key comparisons are required to find the string **`"Gloin"`** in the tree?

`"Gloin" > "Dwalin"`, **`"Gloin" < "Kili"`** , **`"Gloin" > "Fili"`**, and **`"Gloin" == "Gloin"`**

2. Calculating the height of a binary tree

```
/*
 * Function: height
 * Usage: int h = height(tree);
 * ----------------------------
 * Returns the height of the tree, which is defined to be the length
 * of the longest path from the root to a leaf.
 */

int height(BSTNode *tree) {
   if (tree == NULL) {
      return -1;
   } else {
      return max(height(tree->left), height(tree->right)) + 1;
   }
}
```

3. Checking whether a tree is balanced

The simple coding looks like this:

```
/*
 * Function: isBalanced
 * Usage: if (isBalanced(tree)) . . .
 * ----------------------------------
 * Determines whether the tree is balanced, which requires both
 * of the following conditions:
 * 1. The heights of the left and right subtree differ by at most 1.
 * 2. The left and right subtrees are themselves balanced.
 */

bool isBalanced(BSTNode *tree) {
   if (tree == NULL) {
      return true;
   } else {
      return abs(height(tree->left) - height(tree->right)) <= 1
          && isBalanced(tree->left) && isBalanced(tree->right);
   }
}
```

This method, however, requires quadratic time because each subtree will be scanned over and over again as you go up the chain. To fix this problem, you can use any of several strategies. The following code uses a helper function that calculates the height and balance status at the same time (the balance status is returned as the value of the function, and the height is returned through a reference parameter):

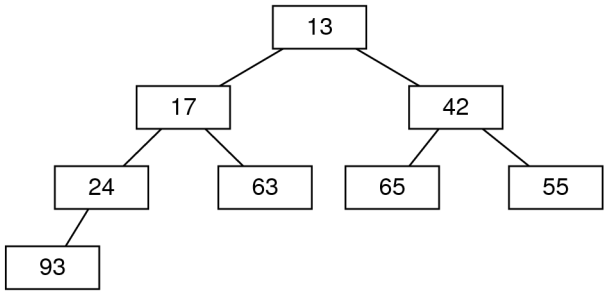
```
bool isBalanced(BSTNode *tree) {
   int height;
   return checkBalance(tree, height);
}

/*
 * Function: checkBalance
 * Usage: (not called by the client)
 * ----------------------------------
 * This function computes two properties of the tree simultaneously.
 * The result of the function itself is a bool indicating whether
 * the tree is balanced.  If it is balanced, the height of the tree
 * is returned in the reference parameter height, so that it can be
 * used in subsequent calculation.  Note that the height value is
 * not guaranteed to be correct if the tree is unbalanced.
 */

bool checkBalance(BSTNode *tree, int & height) {
   if (tree == NULL) {
      height = -1;
      return true;
   }
   int leftHeight, rightHeight;
   if (!checkBalance(tree->left, leftHeight)) return false;
   if (!checkBalance(tree->right, rightHeight)) return false;
   if (abs(leftHeight - rightHeight) > 1) return false;
   height = max(leftHeight, rightHeight) + 1;
   return true;
}
```

Problem 4

4a.

	0	1	2	3	4	5	6	7
after adding 63	63							
after adding 24	24	63						
after adding 55	24	63	55					
after adding 93	24	63	55	93				
after adding 17	17	24	55	93	63			
after adding 65	17	24	55	93	63	65		
after adding 42	17	24	42	93	63	65	55	
after adding 13	13	17	42	24	63	65	55	93

4b.



4c.

	0	1	2	3	4	5	6	7
after removing 13	17	24	42	93	63	65	55	
after removing 17	24	55	42	93	63	65		
after removing 24	42	55	65	93	63			
after removing 42	55	63	65	93				
after removing 55	63	93	65					
after removing 63	65	93						
after removing 65	93							
after removing 93								