CS106B
Spring 2016 Cynthia Lee

Section 5 (Week 6) - SOLUTION

1. **height**.

```
int BinaryTree::height() {
    return height(root);
}

int BinaryTree::height(const TreeNode*& node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + max(height(node->left), height(node->right));
    }
}
```

2. **isBST**.

```
bool BinaryTree::isBST() {
    TreeNode* prev = NULL;
    return isBST(root, prev);
}

// An in-order walk of the tree, storing the last visited node in
// 'prev'
bool BinaryTree::isBST(const TreeNode*& node, TreeNode*& prev) {
    if (node == NULL) {
        return true;
    } else if (!isBST(node->left, prev) ||
               (prev && node->data <= prev->data)) {
        return false;
    } else {
        prev = node;
        return isBST(node->right, prev);
    }
}
```

3. **limitPathSum**.

```
void BinaryTree::limitPathSum(int max) {
    limPathSum(root, max, 0);
}

void BinaryTree::limPathSum(TreeNode*& node, int max, int sum) {
    if (node != NULL) {
        sum += node->data;

        if (sum > max) {
            deleteTree(node); // frees subtree rooted at node
            node = NULL;
        } else {
            limPathSum(node->left,  max, sum);
            limPathSum(node->right, max, sum);
        }
    }
}
```

4. **isBalanced**.

```
bool BinaryTree::isBalanced() {
    return isBalanced(root);
}

bool BinaryTree::isBalanced(const TreeNode*& node) {
    if (node == NULL) {
        return true;
    } else if (!isBalanced(node->left) ||
               !isBalanced(node->right)) {
        return false;
    } else {
        int leftHeight  = height(node->left);
        int rightHeight = height(node->right);
        return abs(leftHeight - rightHeight) <= 1;
    }
}
```

5. **completeToLevel**.

```
void BinaryTree::completeToLevel(int k) {
    if (k < 1) {
        throw k;
    }
    completeToLevel(root, k, 1);
}

void BinaryTree::completeToLevel(TreeNode*& node, int k, int level) {
     if (level <= k) {
         if (node == NULL) {
             node = new TreeNode(-1);
         }
         completeToLevel(node->left, k, level + 1);
         completeToLevel(node->right, k, level + 1);
     }
 }
```

6. **countLeftNodes**.

```
int BinaryTree::countLeftNodes()  {
    return countLeftNodes(root);
}
int BinaryTree::countLeftNodes(TreeNode* node) {
    if (node == NULL) {
        return 0;
    } else if (node->left == NULL) {
        return countLeftNodes(node->right);
    } else {
        return 1 + countLeftNodes(node->left)
                 + countLeftNodes(node->right);
    }
}
```

7. **Traversals.**

a)		b)		c)	
Pre-order:	3 5 1 2 4 6	Pre-order:	19 47 23 -2 55 63 94 28	Pre-order:	2 1 7 4 3 5 6 9 8
In-order:	1 5 3 4 2 6	In-order:	23 47 55 -2 19 63 94 28	In-order:	2 3 4 5 7 1 6 8 9
Post-order:	1 5 4 6 2 3	Post-order:	23 55 -2 47 28 94 63 19	Post-order:	3 5 4 7 8 9 6 1 2