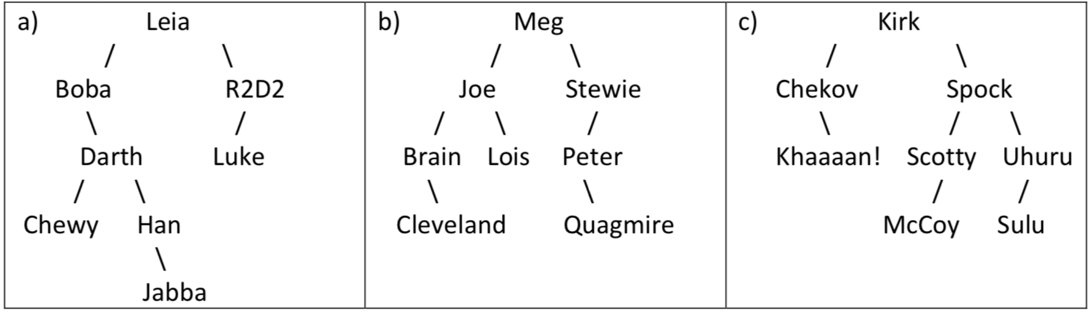
Section #7

Tyler Conklin and Kate Rydberg (Based on handouts by previous CS106B instructors and TAs, especially those of Marty Stepp and Julie Zelenski.)

Trees: Solutions

1. Binary Search Tree Insertion



2. Height

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

3. Find Min

```
int findMin(TreeNode* node) {
    while (node->left != nullptr) {
        node = node->left;
    }
    return node->data;
}
```

Note: if the tree is not a binary search tree, you would have to look at the entire tree to find the minimum value (rather than just going all the way down the left branch).

4. Is Balanced

```
bool isBalanced(TreeNode* node) {
    if (node == nullptr) {
        return true;
    } else if (!isBalanced(node->left) || !isBalanced(node->right)) {
        return false;
    } else {
        // use our 'height' solution from a previous problem
        return abs(height(node->left) - height(node->right)) <= 1;
    }
}
```

5. Is BST

```
bool isBST(TreeNode* node) {
    return isBSTHelper(node, nullptr);
}

// perform in-order walk of tree, comparing current node to last-seen ('prev') node
bool isBSTHelper(TreeNode* node, TreeNode* prev) {
    if (node == nullptr) {
        return true;
    } else if (!isBSTHelper(node->left, prev)) {
        // check left
        return false;
    } else if (prev != nullptr && node->data <= prev->data) {
        // check current node
        return false;
    } else {
        // check right
        return isBSTHelper(node->right, node); // (this node is prev now)
    }
}
```

6. Remove Leaves

```
void removeLeaves(TreeNode*& node) {
    if (node != nullptr) {
        if (node->left == nullptr && node->right == nullptr) {
            delete node;
            node = nullptr; // you can do this since node is passed by reference!
        } else {
            removeLeaves(node->left);
            removeLeaves(node->right);
        }
    }
}
```