CS 106B Section 7 (Week 8) Solutions

1. 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

2. BST Insertion

a)

```
              Leia
            /      \
       Boba          R2D2
         \           /
          Darth     Luke
         /     \
     Chewy      Han
                  \
                   Jabba
```

b)

```
              Meg
            /      \
          Joe        Stewie
         /   \        /
    Brian    Lois   Peter
       \               \
      Cleveland      Quagmire
```

c)

```
              Kirk
            /      \
       Chekov        Spock
         \          /     \
       Khaaan!   Scotty    Uhuru
                  /          /
               McCoy       Sulu
```

3. height

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

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

4. 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);
    }
}
```

CS 106B Section 7 (Week 8) Solutions

5. isBalanced

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

bool BinaryTree::isBalanced(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;
    }
}
```

6. removeLeaves

```
void BinaryTree::removeLeaves() {
    removeLeaves(root);
}

void BinaryTree::removeLeaves(TreeNode*& node) {
    if (node != NULL) {
        if (node->left == NULL && node->right == NULL) {
            delete node;
            node = NULL;
        } else {
            removeLeaves(node->left);
            removeLeaves(node->right);
        }
    }
}
```

7. 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);
    }
}
```

CS 106B Section 7 (Week 8) Solutions

8. Pointer tracing

```
0: 0, 0
1: 1, 10
2: 2, 20
3: 3, 30
4: 4, 40
0: 137, 0
1: 137, 10
2: 137, 20
3: 137, 30
4: 137, 40
0: 137, 0
1: 137, 10
2: 137, 20
3: 137, 30
4: 137, 40
```

Remember that when passing a pointer to a function, *the pointer is passed by value!* This means that you can change the contents of the array being pointed at, because those elements aren't copied when the function is called. On the other hand, if you change *which* array is pointed at, the change does not persist outside the function because you have only changed the copy of the pointer, not the original pointer itself.

9. Hashing (part 1)

hash1 is valid, but not good because everything will get hashed to the same bucket.
hash2 is not valid, because “A” and “a” are equal, but will have different hash values.
hash3 is valid and good.
hash4 is not valid, because equal strings might not give the same hash value.

10. Hashing (part 2)

```
Bucket:

0: baggage (30)
1: badcab (13)
2: feed (20) -> deadbeef (32)
3: cafe (15) -> cabbage (21)
4: null
5: null
```