# Week 7 Section

This week's section handout has practice with trees and hashing. Follow along at Code StepByStep:
https://codestepbystep.com/problemset/view?id=32

For this section handout, we use the BinaryTreeNode struct below:

```
struct BinaryTreeNode {
   int data;
   BinaryTreeNode* left;
   BinaryTreeNode* right;
   ...
}
```

---

## 1. Tree Traversal (traversalsB on CSBS)

Write the elements of the tree below i the order they would be seen by a pre-order, in-order, and post-order traversal.

```
  ______42________
 /                \
15___          _____9
     \        /      \
      27    86__      3
     /          \      \
   48            12     39
                /
               5
```

---

## 2. CountLeftNodes

Write a function named **`countLeftNodes`** that accepts a pointer to the root of a binary tree of integers. Your function should return the number of left children in the tree. A left child is a node that appears as the root of the left-hand subtree of another node.

---

## 3. isBST

Write a member function named **`isBST`** that could be added to the `BinaryTree` class. Your function should return whether or not a binary tree is arranged in valid binary search tree (BST) order. The empty tree is a BST by definition.

---

---

*Thanks to Marty Stepp and other CS106B and X instructors and TAs for contributing problems on this handout.*

## 4. swapChildrenAtLevel

Write a function named **`swapChildrenAtLevel`** that manipulates a binary tree. Your function should accept two parameters: a reference to a pointer to the root of a tree, and an integer *k*, and should swap the left and right children of all nodes at level *k*. In other words, after your function is run, any node at level (*k*+1) that used to be its parent's left child should now be its parent's right child and vice versa. For this problem, the overall root of a tree is defined to be at level 1, its children are at level 2, etc.

---

## 5. Hash Functions

Let's say we have a class `StRiNg` where two `StRiNgs` are considered equal if they are equal, ignoring upper and lower case. Other than that, they are the same as normal strings. Which of the following functions are legal hash functions for `StRiNgs`? Which functions are good hash functions?

```
int hash1(StRiNg& s) {
 return 0;
}
```

```
int hash2(StRiNg& s) {
  return (int) &s;
}
```

```
int hash3(StRiNg& s) {
  int sum = 0;
  for (int i = 0;
     i < s.length(); i++) {
    sum += s[i];
  }
  return sum;
}
```

```
int hash4(StRiNg& s) {
  int product = 1;
  for (int i = 0; i < s.length(); i++) {
    product *= tolower(s[i]);
  }
  return product;
}
```

---