ECE220 Computer Systems and Programming

Lab 11

1 After this week's lectures, you should be able to...

1. Understand tree terminologies such as root, internal/external node, parent, child, height, etc.
2. Describe the defining characteristics of a binary search tree.
3. Differentiate the sequence of access for pre-order, in-order and post-order traversal of a given binary(search) tree.
4. Implement basic tree functions such as the three traversals, search, insert, find height, etc.
5. Implement tree traversal subroutine in LC-3, with emphasis on the recursive caller build up and tear down, and accessing struct members in LC-3.

2 After today's lab, you should be able to...

1. Modify traversal functions for more advanced tree operations.

3 Exercises

1. Implement a function that returns 1 if node is the right child of its parent, and 0 otherwise.

```
typedef struct NODE {
  int value;
  struct NODE* parent;
  struct NODE* left;
  struct NODE* right;
}node_t;
int is_right_child(node_t* node){
    return node->parent->right == node;


}
```

2. Modify the two simple tree operations below to achieve the alternative tasks specified in the comments. Assume the same node definition as exercise 1. There are many online learning resources about trees. The goal of this exercise is to give you an idea on how to modify basic tree operations to suit tasks in the MP.

```
1  // Modify this function to return the sum of node values, instead of a count.
2  int count(node_t* node){
3      if(node == NULL) return 0;
4      return 1 + count(node->left) + count(node->right);
5  }    return node->value + count(node->left) + count(node->right);
6  // Modify this function to only print leaf nodes, instead of all nodes.
7  void inorder(node_t* node)
8  {
9        if (node == NULL) return;
10       inorder(node->left);
11       printf("%d ", node->value);
12       inorder(node->right);
13 }
```

between line 10 and 11
if(node->left == NULL && node->right==NULL)