COP 3502 Recitation Sheet: AVL Trees, Tries

Directions: Each of these questions is from either a past Foundation Exam or one of my past exams. They were created to be written on paper. However, the trie questions can be coded on a computer and tested. I strongly recommend first coding on paper, but then transferring to the computer and testing the function until you are convinced it works.

1) Consider inserting the following items into an initially empty AVL tree, in the order shown. Show the state of the tree after each insertion completes.

10, 6, 3, 20, 80, 15, 60, 100, 18, 90

2) Draw an AVL tree **of integers** and designate a single node in the AVL tree such that, if that node were to be deleted, two rebalance operations (not one double rotation, but two separate operations at two different nodes) would occur. Clearly label the node to delete which would precipitate those operations and show the result of deleting that node. (Thus, you should have two drawings, a before drawing of the original tree with the node to be deleted clearly designated, and an after drawing showing what the tree looks like after the node is deleted and goes through 2 rebalance operations.)

3) Write a function that takes the root of a trie (*root*) and returns the number of strings in that trie that **end** with the letter *q*. The *count* member of the node struct indicates how many occurrences of a particular string have been inserted into the trie. So, a string can be represented in the trie multiple times. If a string ending in *q* occurs multiple times in the trie, all occurrences of that string should be included in the value returned by this function.

The node struct and function signature are given below. You must write your solution in a **single** function. You cannot write any helper functions.

```
typedef struct TrieNode
{
   // Pointers to the child nodes (26 total).
   struct TrieNode *children[26];

   // Indicates how many occurrences of a particular string are
contained
   // in this trie. If none, this is set to zero (0).
   int count;
} TrieNode;

int ends_with_q_count(TrieNode *root);
```

4) The word “intention” is such that four of its prefixes, “i”, “in”, “intent” and “intention” are words themselves. Write a function that takes in a pointer to the root of a trie storing a dictionary of words and returns the maximum number of words that are prefixes of a single word. Use the struct definition and function prototype given below.

```
typedef struct TrieNode {
    struct TrieNode *children[26];
    int flag; // 1 if the string is in the trie, 0 otherwise
} TrieNode;

int maxNumPrefixWords(TrieNode* root);
```

5) It's often useful to know how many words start with a particular prefix. Given a trie that stores a dictionary of valid words (**lowercase letters only**) as well as a prefix string, write a **non-recursive** function that calculates the number of words that begin with that prefix. To aid you in your solution, the struct that stores a trie node will not only store whether or not that node represents a word or not, but it will **also** store the total number of words stored within that subtree of the trie in a variable called numwords. You may assume that the TrieNode pointer passed to the function represents the root of the whole trie storing the dictionary of words. You may assume that root is NOT NULL and prefix has at least one lowercase letter in it.

```
#include <string.h>

typedef struct TrieNode {
    struct TrieNode *children[26];
    int flag; // 1 if the string is in the trie, 0 otherwise
    int numwords; // the total # of words stored in this sub-trie.
} TrieNode;

int numWordsWithPrefix(TrieNode* root, char* prefix);
```