COP 3502 Recitation Sheet: Trees

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, each of these 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.

Each of the following questions asks you to write a recursive function.

1) Write a function that returns 1, if the two binary trees pointed to by root1 and root2, respectively are equal trees, and 0 otherwise.. For two trees to be equal, their structure must be identical AND the values stored at each corresponding node must be the same. (Hint: Thus, for non-empty trees, in order for two trees to be equal, the values stored at both roots must be the same. In addition, both the two left trees and the two right trees must ALSO be equal trees.) Use the struct shown below.

```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} btreenode;

int equal(btreenode* root1, btreenode* root2);
```

2) Write a function that takes in a pointer to a binary tree and an integer called add, and adds that integer to the value stored at each node in the tree. Use the same struct as question #1. The function prototype is given below.

```
void addConstant(btreenode* root, int add);
```

3) Write a function named *fsl()* (which stands for "**f**ind **s**mallest **l**eaf") that takes a pointer to the root of a binary tree as its only argument and returns the value of the smallest **leaf** node in the tree. Note that the tree passed to your function will **not** necessarily be a binary **search** tree. If the pointer root is NULL, fsl should return INT_MAX, which is defined below.

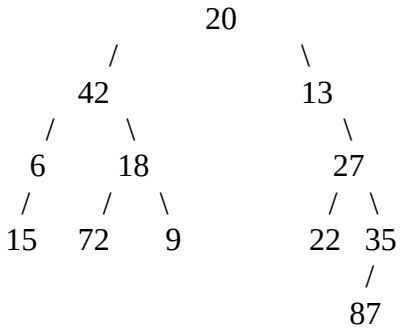
You **cannot** write any helper functions for this problem. You must complete all of your work in a single function. The function signature and node struct are given below.

```
#define INT_MAX 2147483647

typedef struct node {
    int data;
    struct node *left;
    struct node *right;
} node;

int fsl(node *root);
```

4) Consider a function that takes in a pointer to a binary tree node and returns a pointer to a binary tree node defined below:

```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} btreenode;

btreenode* somefunction(btreenode* root) {
    if (root == NULL) return NULL;
    somefunction(root->left);
    somefunction(root->right);
    btreenode* tmp = root->left;
    root->left = root->right;
    root->right = tmp;
    return root;
}
```

Let the pointer tree point to the root node of the tree depicted below:



If the line of code tree = somefunction(tree) were executed, draw a picture of the resulting binary tree that the pointer tree would point to.

5) Write a function named *find_below*() that takes a pointer to the root of a binary tree (*root*) and an integer value (*val*) and returns the greatest value in the tree that is strictly less than *val*. If no such value exists, simply return *val* itself. Note that the tree passed to your function will **not** necessarily be a binary **search** tree; it's just a regular binary tree.

For example:

```
        18           find_below(root, 196) would return 22
       /  \          find_below(root, 1) would return 1
      7    4         find_below(root, 4) would return 1
     / \             find_below(root, 22) would return 18
    1  22            find_below(root, 20) would return 18
         \           find_below(root, 8) would return 7
          8          find_below(root, -23) would return -23
```

You must write your solution in a **single** function. You cannot write any helper functions.

The function signature and node struct are given below.

```
typedef struct node
{
   int data;
   struct node *left;
   struct node *right;
} node;

int find_below(node *root, int val);
```