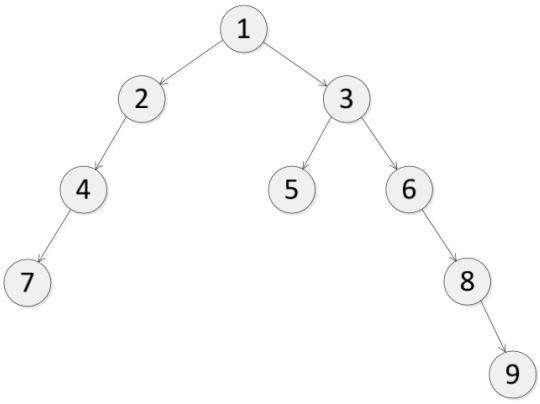
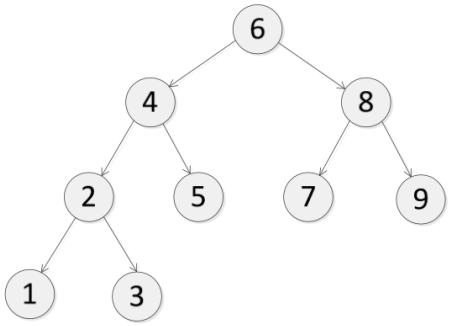
Homework #6

In this assignment you are asked to implement a variety of functions that operate on binary trees (the binary tree implementation from the book). You will be asked to test these functions on the following two trees (element data type is int):

Tree #1



Tree #2



For parts a through g, implement the given function and demonstrate the function working with the two trees provided earlier in this document:

a) **(1 point)** int count_leaves (BiTree *tree);

Returns the number of leaf nodes in the tree.

hw6 > C hw6.c > ...

```c
#include "bitree.h"
#include <stdio.h>
#include <stdlib.h>

// Recursion to count the leaves for each node
int count_recursion(BiTreeNode *node) {
    if (node == NULL) {                 // if not leaf node
        return 0;                       // return 0 for False, (also good for counting)
    }
    if (bitree_left(node) == NULL &&    // if the node has no left or right children
        bitree_right(node) == NULL) {   // In other words, if the node is a leaf node
        return 1;                       // return 1 for True, (also good for counting)
    }
    return count_recursion(node->left) + count_recursion(node->right);
}

int count_leaves (BiTree *tree) {
    return count_recursion(tree->root);
}
```

```c
// Ceates an integer on the heap
int* create_int(int value) {
    int* new_int = (int*)malloc(sizeof(int));
    *new_int = value;
    return new_int;
}

// Free function for bitree_init
void free_int(void* data) {
    free(data);
}

```

```c
int main() {
    // Initialize the 2 trees given in the assignment
    BiTree tree1, tree2;
    BiTreeNode *node;
    bitree_init(&tree1, free_int);
    bitree_init(&tree2, free_int);

    // Insert for nodes tree1
    bitree_ins_left(&tree1, NULL, create_int(1));
    node = tree1.root;
    bitree_ins_left(&tree1, node, create_int(2));
    bitree_ins_left(&tree1, node->left, create_int(4));
    bitree_ins_left(&tree1, node->left->left, create_int(7));
    bitree_ins_right(&tree1, node, create_int(3));
    bitree_ins_left(&tree1, node->right, create_int(5));
    bitree_ins_right(&tree1, node->right, create_int(6));
    bitree_ins_right(&tree1, node->right->right, create_int(8));
    bitree_ins_right(&tree1, node->right->right->right, create_int(9));

    // Insert for nodes tree2
    bitree_ins_left(&tree2, NULL, create_int(6));
    node = tree2.root;
    bitree_ins_left(&tree2, node, create_int(4));
    bitree_ins_left(&tree2, node->left, create_int(2));
    bitree_ins_right(&tree2, node->left, create_int(5));
    bitree_ins_left(&tree2, node->left->left, create_int(1));
    bitree_ins_right(&tree2, node->left->left, create_int(3));
    bitree_ins_right(&tree2, node, create_int(8));
    bitree_ins_left(&tree2, node->right, create_int(7));
    bitree_ins_right(&tree2, node->right, create_int(9));

    // count leaves
    int leaves1 = count_leaves (&tree1);
    int leaves2 = count_leaves (&tree2);
    printf("Number of leaves in tree1: %d\n", leaves1);
    printf("Number of leaves in tree2: %d\n", leaves2);

    // Destroy the trees to clean up
    bitree_destroy(&tree1);
    bitree_destroy(&tree2);

    return 0;
}
```

```
~/Desktop/DSA/hw6 main* ❯ ./hw6
Number of leaves in tree1: 3
Number of leaves in tree2: 5
```

b) **(1 point)** int count_non_leaves (BiTree *tree);
Returns the number of non-leaf nodes in the tree.

```c
// Recursion to count number of non-leaf nodes in the tree
int non_leaves_recursion (BiTreeNode *node) {
    if (node == NULL) {                                 // if node is empty
        return 0;                                       // return 0
    }
    if (node->left != NULL || node->right != NULL){     // if either child node isn't empty -> it's not a leaf node
        return 1 + non_leaves_recursion(node->left) + non_leaves_recursion(node->right);
    }
    return 0;
}

int count_non_leaves (BiTree *tree) {
    return non_leaves_recursion(tree->root);
}
```

```
~/Desktop/DSA/hw6 main* ❯ ./hw6
Number of leaves in tree1: 3
Number of leaves in tree2: 5
Number of non-leaf nodes in tree1: 6
Number of non-leaf nodes in tree2: 4
```

c) **(1 point)** int get_height (BiTree *tree);
Returns the height of the tree.

```c
// Returns the height of the tree
int get_height_recursion (BiTreeNode *node) {
    if (node == NULL) {
        return 0;
    }

    // Recursion to separately get heights of left side of binary tree as well as right side
    int left_height = get_height_recursion(node->left);
    int right_height = get_height_recursion(node->right);

    // if left > right, return left; else return right
    return 1 + (left_height > right_height ? left_height : right_height);   // height + 1 (for root node)
}

int get_height(BiTree *tree) {
    return get_height_recursion(tree->root);
}
```

```
~/Desktop/DSA/hw6 main* ❯ ./hw6
Number of leaves in tree1: 3
Number of leaves in tree2: 5
Number of non-leaf nodes in tree1: 6
Number of non-leaf nodes in tree2: 4
Height of tree1: 5
Height of tree2: 4
```

d) **(1 point)** void print_pre_order (BiTree *tree,
void (*print) (const void *data))

Prints the elements of the tree to stdout using a pre-order traversal. The print parameter should contain the logic to print the data held in each node in the tree.

```c
// print pre order
void print_pre_order_recursion (BiTreeNode *node, void (*print)(const void *data)) {
    if (node == NULL) {
        return;
    }

    // root node
    print(bitree_data(node));

    // recursion to print left and right nodes in order after the root node (post-traversally)
    print_pre_order_recursion(bitree_left(node), print);
    print_pre_order_recursion(bitree_right(node), print);
}

void print_pre_order (BiTree *tree, void (*print)(const void*data)) {
    print_pre_order_recursion(tree->root, print);
}

void print_node(const void *data) {
    printf("%d ", *(int *)data);
}
```

```
1 2 4 7 3 5 6 8 9
6 4 2 1 3 5 8 7 9
```

e) **(1 point)** void print_in_order (BiTree *tree,
void (*print) (const void *data))

Prints the elements of the tree to stdout using an in-order traversal. The print parameter should contain the logic to pretin the data held in each node in the tree.

```c
// print in-order (from left to right regardless of height)
void print_in_order_recursion(BiTreeNode *node, void (*print)(const void *data))
    if (node == NULL) {
        return;
    }

    // Recursively traverse the left subtree first
    print_in_order_recursion(bitree_left(node), print);

    // then the root node
    print(bitree_data(node));

    // finally, traverse the right subtree
    print_in_order_recursion(bitree_right(node), print);
}

void print_in_order(BiTree *tree, void (*print)(const void *data)) {
    print_in_order_recursion(tree->root, print);
}
```

```
Print in order
7 4 2 1 5 3 6 8 9
1 2 3 4 5 6 7 8 9
```

f) **(1 point)** void print_post_order (BiTree *tree,
void (*print) (const void *data))

Prints the elements of the tree to stdout using a post-order traversal. The print parameter should contain the logic to print the data held in each node in the tree.

```c
// print post-order
void print_post_order_recursion(BiTreeNode *node, void (*print)(const void *data)) {
    if (node == NULL) {
        return;
    }

    // Recursively traverse the left subtree first
    print_post_order_recursion(bitree_left(node), print);

    // then, traverse the right subtree
    print_post_order_recursion(bitree_right(node), print);

    // finally, the root node
    print(bitree_data(node));
}

void print_post_order(BiTree *tree, void (*print)(const void *data)) {
    print_post_order_recursion(tree->root, print);
}
```

```
Print post order
7 4 2 5 9 8 6 3 1
1 3 2 5 4 7 9 8 6
```

g) **(3 points)** void remove_leaves(BiTree *tree)

Removes all leaf nodes from the tree. Use print_pre_order, print_in_order, or print_post_order after calling remove_leaves to show that remove_leaves successfully removed all leaves.

```c
// Remove Leaves
BiTreeNode *remove_leaves_recursion(BiTreeNode **node){                // use double pointers to directly modify the tree
    if (node == NULL || *node == NULL) {
        return NULL;
    }

    if ((*node)->left == NULL && (*node)->right == NULL) {             // if the current node is a leaf node
        BiTreeNode *temp = *node;                                      // store current node address to another pointer
        *node = NULL;                                                  // remove the node
        free(temp);                                                    // free allocated memory
        return NULL;
    }

    // use recursion to repeat the process for all nodes until no more leaf nodes exist on the tree
    (*node)->left = remove_leaves_recursion(&((*node)->left));
    (*node)->right = remove_leaves_recursion(&((*node)->right));

    // Remove nodes again after the first set of leaves has been removed
    if ((*node)->left == NULL && (*node)->right == NULL) {
        BiTreeNode *temp = *node;
        *node = NULL;
        free(temp);
        return NULL;
    }

    return *node;
}

void remove_leaves(BiTree *tree){
    tree->root  = remove_leaves_recursion(&(tree->root));
}
```

```c
    // remove leaves
    printf("Remove all leaves of the trees\n");
    remove_leaves(&tree1);
    remove_leaves(&tree2);

    // print pre order
    printf("Print pre order\n");
    print_pre_order(&tree1, print_node);
    printf("\n");
    print_pre_order(&tree2, print_node);
    printf("\n");

    // print in order
    printf("Print in order\n");
    print_in_order(&tree1, print_node);
    printf("\n");
    print_in_order(&tree2, print_node);
    printf("\n");

    // print post order
    printf("Print post order\n");
    print_post_order(&tree1, print_node);
    printf("\n");
    print_post_order(&tree2, print_node);
    printf("\n");
```

```
~/Desktop/DSA/hw6 main* ❯ clang -o hw6 hw6.c bitree.c
~/Desktop/DSA/hw6 main* ❯ ./hw6
Number of leaves in tree1: 3
Number of leaves in tree2: 5
Number of non-leaf nodes in tree1: 6
Number of non-leaf nodes in tree2: 4
Height of tree1: 5
Height of tree2: 4
Print pre order
1 2 4 7 3 5 6 8 9
6 4 2 1 3 5 8 7 9
Print in order
7 4 2 1 5 3 6 8 9
1 2 3 4 5 6 7 8 9
Print post order
7 4 2 5 9 8 6 3 1
1 3 2 5 4 7 9 8 6
Remove all leaves of the trees
Print pre order


Print in order


Print post order


~/Desktop/DSA/hw6 main* ❯
```