**a. to count the number of nodes in a binary tree**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

//Standard Binary node structure

struct Node {

int data;

struct node\* left;

struct node\* right;

};

struct Node\* newNode (int data){

struct Node\* node = (struct Node\*)

malloc(sizeof(struct Node));

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

};

int nodeTotal = 0;

int countNodes (node\* node){

//If not null reference, count node in total

if (node != NULL)

nodeTotal++;

//Traverse rest of tree to find other existing nodes

return countNodes(node->left)

&& countNodes(node->right);

};

int main (){

struct node\* root = newNode(5);

root->left = newNode(3);

root->left->left = newNode(2);

root->left->right = newNode(4);

root->right = newNode(7);

root->right->left = newNode(6);

root->right->right = newNode(8);

cout << countNodes(root) << endl;

return 0;

};

Addiitonal functions can be added to A and executed as needed.

**b. to count the number of leaves**

int leafTotal = 0;

int countLeaves (node\* node){

//If both left and right pointers are null, node is leaf.

if (node->left = NULL && node->right = NULL)

leafTotal++;

//Call recursive search on both left and right pointers.

return countLeaves(node->right) && countLeaves(node->left);

};

**c. to count the number of right children**

int rightCount = 0;

int countRightChildren (node\* node){

//If right pointer is not null, a child exists, count it

if (node->right != NULL)

rightCount++;

//Traverse rest of tree to find all right children

return countRightChildred(node->left)

&& countRightChildren(node->right);

};

**d. to find the height of the tree**

using namespace std

//use tree as argument

int Height(tree \*p)

{

//initialize variables for left, right, and highest branch

int rHeight = 0;

int lHeight = 0;

int highest = 0;

//if the tree is empty return 0

if (p == NULL)

{

return height;

}

// if the left or right node is not empty count it to its variable

else if (node->left != NULL)

{

lHeight++;

}

else if (node->right != NULL)

{

rHeight++;

}

//get the highest branch of the tree and add the root node to get the height and return

highest = (1 + max(lHeight, rHeight);

return highest;

}

**e. to delete all leaves from a binary tree**

int deleteLeaves (node\* node){

if (node->left = NULL && node->right = NULL)

delete node;

return deleteLeaves(node->right)

&& deleteLeaves(node->left);

};