Q1

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data = data;

this->left = NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root;

Tree(){

root = NULL;

}

// Insertion of the node to the tree

Node\* insert(Node\* node, int val){

if(node == NULL){

return new Node(val);

}

if(val < node->data){

node->left = insert(node->left,val);

}else if(val > node->data){

node->right = insert(node->right,val);

}

return node;

}

// In order traversal

void inOrderTraversal(Node\* node){

if (node == NULL) {

return;

}

inOrderTraversal(node->left);

cout<< node->data<< " ";

inOrderTraversal(node->right);

}

};

int main(){

Tree tree;

tree.root =tree.insert(tree.root,1);

tree.insert(tree.root,2);

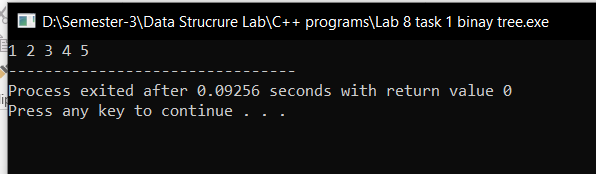
tree.insert(tree.root,3);

tree.insert(tree.root,4);

tree.insert(tree.root,5);

tree.inOrderTraversal(tree.root);

}



Q2 And Q3

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data = data;

this->left = NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root;

int maxh=0;

Tree(){

root = NULL;

}

//Inserting the node to the tree

Node\* insert(Node\* node, int val){

if(node == NULL){

return new Node(val);

}

if(val < node->data){

node->left = insert(node->left,val);

}else if(val > node->data){

node->right = insert(node->right,val);

}

return node;

}

//Get the height of the tree

int getHeight(Node\* root){

if(root==NULL){

return 0;

}

int left = getHeight(root->left);

int right = getHeight(root->right);

int maxh = max(left,right)+1;

return maxh;

}

// Find the Successor

Node\* successor(Node\* root){

while(root->left!=NULL){

root = root->left;

}

return root;

}

// Deletion in the binary tree

Node\* deleteNode(Node\* root, int value){

if(root==NULL){

return root;

}

if(value<root->data){

root->left = deleteNode(root->left,value);

}

else if(value > root->data){

root->right = deleteNode(root->right,value);

}

else{

// Node with 1 or 2 child

if(root->left==NULL){

Node\* temp = root->right;

delete root;

return temp;

}

else if(root->right==NULL){

Node\* temp = root->left;

delete root;

return temp;

}

// Node with two children

Node\* temp = successor(root->right);

root->data = temp->data;

root->right=deleteNode(root->right,temp->data);

}

return root;

}

// In order traversal

void inOrderTraversal(Node\* node){

if (node == NULL) {

return;

}

inOrderTraversal(node->left);

cout<< node->data<< " ";

inOrderTraversal(node->right);

}

int level(Node\* root, int value){

if(root==NULL){

return -1;

}

if(root->data==value){

return maxh;

}

int left = level(root->left,value);

int right = level(root->right,value);

maxh = max(left,right)+1;

return maxh;

}

//Find fun

void search(Node\* root, int value){

if(root==NULL){

return;

}

else if(value == root->data){

cout <<"\nFound "<<root->data<<" At Height "<<level(root,value)<<endl;

}else{

insert(root,value);

}

search(root->left,value);

search(root->right,value);

}

};

int main(){

Tree tree;

tree.root =tree.insert(tree.root,1);

tree.insert(tree.root,2);

tree.insert(tree.root,3);

tree.insert(tree.root,4);

tree.insert(tree.root,5);

tree.insert(tree.root,6);

tree.insert(tree.root,7);

tree.insert(tree.root,8);

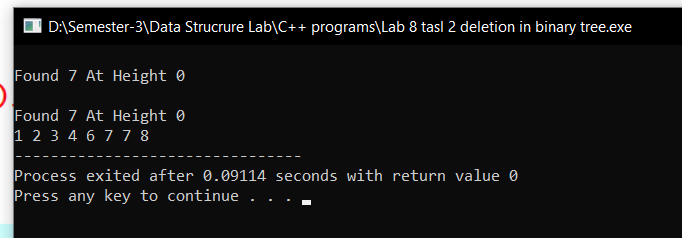
// cout<<"Height is "<<tree.getHeight(tree.root)<<endl;

tree.search(tree.root,7);

tree.deleteNode(tree.root,5);

tree.inOrderTraversal(tree.root);

}



Q 4

//osmn

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data = data;

this->left = NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root;

Tree(){

root=NULL;

}

// Insertion of the node

Node\* insert(Node\* root, int value){

if(root==NULL){

return new Node(value);

}

if(value<root->data){

root->left = insert(root->left,value);

}else if(value > root->data){

root->right = insert(root->right,value);

}

return root;

}

// In order Traversal

void inOrder(Node\* root){

if(root == NULL){

return ;

}

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

};

int main(){

int n;

int sel;

int data;

Tree tree;

tree.root =tree.insert(tree.root,7);

tree.insert(tree.root,5);

tree.inOrder(tree.root);

cout<<"\nHow many numbers you want to enter"<<endl;

cin>>n;

for(int i=0; i<n;i++){

cout<<"What do you want ceil (1) or Floor(0)"<<endl;

cin>>sel;

cout<<"Enter the value"<<endl;

cin>>data;

if(sel== 0){

data = data+1;

tree.insert(tree.root,data);

cout<<"Done"<<endl;

}else if(sel == 1){

data = data-1;

tree.insert(tree.root,data);

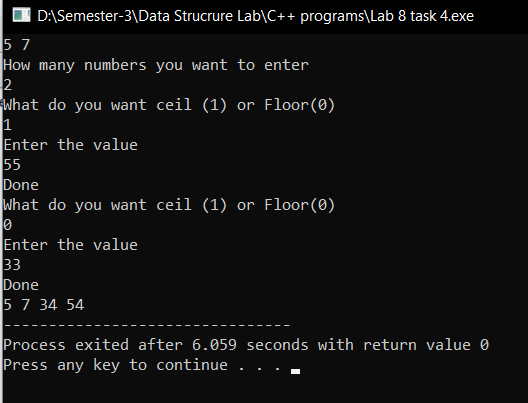
cout<<"Done"<<endl;

}

}

tree.inOrder(tree.root);

}



Q5

//osmn

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data = data;

this->left= NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root;

Tree(){

root = NULL;

}

Node\* insert(Node\* root, int data){

if(root==NULL){

return new Node(data);

}

if(data<root->data){

root->left = insert(root->left,data);

} else if(data > root->data){

root->right = insert(root->right,data);

}else{

root->right = insert(root->right, data);

}

return root;

}

void inOrder(Node\* root){

if(root==NULL){

return ;

}

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

// join

void join(Node\* t,Node\* &t2){

if(t==NULL){

return ;

}

join(t->left,t2);

t2 =insert(t2,t->data);

join(t->right,t2);

}

// Merge

void merge(Tree\* t1, Tree\* t2){

join(t1->root,t2->root);

}

};

int main(){

Tree tree1;

tree1.root = tree1.insert(tree1.root,5);

tree1.insert(tree1.root,3);

tree1.insert(tree1.root,2);

tree1.insert(tree1.root,6);

tree1.insert(tree1.root,4);

// tree1.insert(tree1.root,4);

Tree tree2;

tree2.root = tree2.insert(tree2.root,2);

tree2.insert(tree2.root,1);

tree2.insert(tree2.root,3);

tree2.insert(tree2.root,7);

tree2.insert(tree2.root,6);

tree1.merge(&tree1, &tree2);

// tree1.inOrder(tree1.root);

tree2.inOrder(tree2.root);

}

