AHMAD RAZA

[Company name]  [Company address]

LAB TASK 7

**TASK NO 1:**

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left;

node\* right;

};

class BST\_Operation

{

public:

int count; //count leaf node

int size;

node\* root;

BST\_Operation()

{

count = 0;

size = 0;

root = NULL;

}

node\* insert(node\* root, int val);

int totalnumber();

int degree(int n);

int dept(node\* temp);

int level(node\* temp,int data,int Level);

bool isleaf(node\* temp);

bool isparent(node\* temp);

bool ischild(node\* temp);

bool isDescendent(node\* temp);

bool isSuccessor(node \* temp);

bool issibling(node\* temp, node\* left, node\* right);

void Display(node\* temp);

void leafNodeCount(node\* temp);

int InternalNodeCount(node \* temp);

node\* search(int val);

};

int main()

{

BST\_Operation Obj;

node\* temp = new node;

temp = NULL;

temp = Obj.insert(temp, 100);

Obj.size++;

temp = Obj.insert(temp, 400);

Obj.size++;

temp = Obj.insert(temp, 500);

Obj.size++;

temp = Obj.insert(temp, 20);

Obj.size++;

temp = Obj.insert(temp, 30);

Obj.size++;

temp = Obj.insert(temp, 10);

Obj.size++;

cout << "Data inserted \n\n";

int x;

Obj.leafNodeCount(Obj.root); //calculate the count of leaf node

a: cout << "PRESS 1 To check the height of the BST\n";

cout << "PRESS 2 To check the level of node \n";

cout << "PRESS 3 To check the Degree of the node\n";

cout << "PRESS 4 To check the total number of nodes\n";

cout << "PRESS 5 To check the isleaf node\n";

cout << "PRESS 6 To check the is nonleaf node\n";

cout << "PRESS 7 To check the if it is parent node\n";

cout << "PRESS 8 To check the if it is child or not \n";

cout << "PRESS 9 To check if it is Descendent\n";

cout << "PRESS 10 To check if it is Successor\n";

cout << "PRESS 11 TO Check if it is sibling or not \n";

cout << "Press 12 To Count the leaf node \n";

cout << "Press 13 to Count the internal node\n";

cout << "Press 14 to exit \n";

cin >> x;

if (x == 1)

{

cout << "The Height of the binary tree is : " << Obj.dept(Obj.root) << "\n";

goto a;

}

if (x == 2)

{

Obj.Display(Obj.root);

cout << endl;

int val;

cout << "Enter the root value to check its level \n";

cin >> val;

cout << "The Level of the root "<<val <<" is : " << Obj.level(Obj.root, val, 0) << "\n";//pass level as 0

goto a;

}

if (x == 3)

{

int n;

n = Obj.count;

cout << "The Degree Of the binary tree is : " << Obj.degree(n)<<"\n";

goto a;

}

if (x == 4)

{

cout << "The total number of nodes is : " << Obj.totalnumber() << "\n";

goto a;

}

if (x == 5)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value to check if it is a leaf node \n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.isleaf(temp1))

cout << "\nLeaf node \n";

else

cout << "\nNot leaf node \n";

goto a;

}

if (x == 6)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value to check if it is non-leaf node \n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.isleaf(temp1))

cout << "\nLeaf node \n";

else

cout << "\nNon leaf node \n";

goto a;

}

if (x == 7)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value of node to check if it is parent node or not\n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.isparent(temp1))

cout << "\nParent node \n";

else

cout << "\nNot Parent node \n";

goto a;

}

if (x == 8)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value of node to check if it is child node or not\n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.ischild(temp1))

cout << "\nChild node \n";

else

cout << "\nNot child node \n";

goto a;

}

if (x == 9)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value of node to check if it is Descendent node or not\n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.isDescendent(temp1))

cout << "\nNode is Descendent \n";

else

cout << "\nNode is not Descendent\n";

goto a;

}

if (x == 10)

{

Obj.Display(Obj.root);

int n;

cout << "Enter the value of node to check if node has Successor node or not\n";

cin >> n;

node\* temp1 = Obj.search(n);

if (Obj.isSuccessor(temp1))

cout << "\nNode has Successor Node \n";

else

cout << "\nNode does not has Successor node\n";

goto a;

}

if (x == 11)

{

int n, x1;

node obj;

Obj.Display(Obj.root);

cout << "Enter the 1st value \n";

cin >>n;

cout << "Enter the 2st value \n";

cin >> x1;

node\* left = Obj.search(n);

node\* right = Obj.search(x1);

if (Obj.issibling(Obj.root,left,right))

cout << "\nBoth are sibling \n";

else

cout << "\nthey are not sibling\n";

goto a;

}

if (x == 12)

{

cout << "The count of the leaf Node Is : " << Obj.count << endl;;

goto a;

}

if(x == 13)

{

int count;

count=Obj.InternalNodeCount(Obj.root);

cout << "The Total Number of internal Node is : " << count<<" \n";

goto a;

}

if (x == 14)

{

exit;

}

if (x > 14 || x < 1)

{

cout << "\nWrong Input\n";

goto a;

}

system("pause");

return 0;

}

void BST\_Operation::leafNodeCount(node \* root) //calulte the leaf node

{

if (root)

{

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

{

count++;

}

else

{

if (root->left)

{

leafNodeCount(root->left);

}

if (root->right)

{

leafNodeCount(root->right);

}

}

}

}

bool BST\_Operation::issibling(node\* temp, node\* left, node\* right) //check if the both node are sibling or not

{

if (temp == NULL)

{

return false;

}

return(temp->left == left && temp->right == right)

|| (temp->left == right && temp->right == left)

|| issibling(temp->left, left, right)

|| issibling(temp->right, left, right);

}

node\* BST\_Operation::search(int val)

{

node \*temp1 = root;

while (temp1 != NULL)

{

if (val > temp1->data)

temp1 = temp1->right;

else if (val < temp1->data)

temp1 = temp1->left;

else

break;

}

return temp1;

}

void BST\_Operation::Display(node\* temp)

{

if (temp == NULL)

{

return;

}

Display(temp->left);

cout << temp->data << "\n";

Display(temp->right);

}

int BST\_Operation::dept(node\* temp) //return the height of the binary tree

{

int h;

if (temp == NULL)

{

return 0;

}

int right = dept(temp->right);

int left = dept(temp->left);

//compare

if (right > left)

{

h=right + 1;//hieght=1+number of edgers on the longest path

}

else

{

h=left + 1;

}

return h;

}

node\* BST\_Operation::insert(node\* temp1, int val)

{

node\* temp = new node;

temp->data = val;

temp->left = NULL;

temp->right = NULL;

if (root == NULL)

{

root = temp;

}

if (temp1 == NULL)

{

temp1 = temp;

return temp1;

}

else if (val > temp1->data)

{

temp1->right = insert(temp1->right, val);

}

else if (val < temp1->data)

{

temp1->left = insert(temp1->left, val);

}

return temp1;

}

int BST\_Operation::level(node\* temp,int data,int Level) //calculate the level of the node

{

if (temp == NULL)

{

return 0;

}

if (temp->data == data)

{

return Level;

}

int lev = level(temp->left, data, Level + 1);

if (lev != 0)

{

return lev;

}

return level(temp->right, data, Level + 1);

}

bool BST\_Operation::isleaf(node\* temp) //check if the node is leaf node or not

{

if (temp == NULL)

{

return false;

}

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

{

return true;

}

return false;

}

bool BST\_Operation::isparent(node\* temp) //check if the node is the parent node or not

{

if (temp == NULL)

{

return false;

}

if (temp->right != NULL || temp->left != NULL)

{

return true;

}

return false;

}

bool BST\_Operation::ischild(node\* temp)//check if the noe is child node or not

{

if (root->data == temp->data)

{

return false;

}

else

{

return true;

}

}

int BST\_Operation::totalnumber() //return the total number of node in a tree

{

return size;

}

bool BST\_Operation::isDescendent(node\* temp) //check if the node is descendent or not

{

if (root->data == temp->data)

{

return false;

}

else

{

return true;

}

}

bool BST\_Operation::isSuccessor(node\* temp) //check if the node is successor or not

{

if (temp->right == NULL)

{

return false;

}

else

{

return true;

}

}

int BST\_Operation::degree(int n) //calculate the degree of the binary tree

{

int deg = 2\*(n-1); //formula to calculate the degree n is a count of leaves node

return deg;

}

int BST\_Operation::InternalNodeCount(node\* temp) //calculate the internal node

{

if (root == NULL)

{

return 0;

}

return 1 + InternalNodeCount(root->left) + InternalNodeCount(root->right);

}

**OUTPUT:**



















