7.0.Program that implements Binary search tree for the following operation: - Insert, Recursive traversal: preorder, postorder, inorder, search, largest node, smallest node, count number of nodes.

#include<iostream> #include<conio.h> using namespace std; class node

{

public:

};

node \*left,\*right; int data;

class binary\_tree

{

public:

node \*root; void insert (int);

void add(int,node\*); void inorder(node\*); void preorder(node\*); void postorder(node\*); void rightorder(node\*); node\* search(int);

void deletenode(int); void minelement(); void maxelement();

binary\_tree() //constructor

{

root=NULL;

}

};

void binary\_tree::insert(int data)

{

if(root!=NULL)

add(data, root);

else

{

}

}

root=new node; root->data=data; root->left=NULL; root->right=NULL;

void binary\_tree::rightorder(node \*temp)

{

}

void binary\_tree::add(int data, node \*temp)

{

if(data < temp->data)

{

if(temp->left!=NULL)

add(data, temp->left);

else

{

temp->left=new node; temp->left->data=data; temp->left->left=NULL; temp->left->right=NULL;

}

}

else if(data >= temp->data)

{

if(temp->right!=NULL)

add(data, temp->right);

else

{

}

}

}

temp->right=new node; temp->right->data=data; temp->right->left=NULL; temp->right->right=NULL;

void binary\_tree::preorder (node\* temp)

{

if (temp!=NULL)

{

cout<<"\n\t"<<temp->data; preorder (temp->left); preorder (temp->right);

}

}

void binary\_tree::inorder(node\* temp)

{

if(temp!=NULL)

{

inorder(temp->left); cout<<"\n\t"<<temp->data; inorder(temp->right);

}

}

void binary\_tree::postorder(node\* temp)

{

if(temp!=NULL)

{

postorder(temp->left); postorder(temp->right); cout<<"\n\t"<<temp->data;

}

}

void binary\_tree::minelement()

{

node \*temp; node \*pre; temp=root;

while(temp!=NULL)

pre=temp; temp=temp->left;

}

cout<<"\n\tMin element in tree is= "<<pre->data;

}

void binary\_tree::maxelement()

{

node \*temp; node \*pre; temp=root; pre=temp;

while(temp!=NULL)

{

pre=temp; temp=temp->right;

}

cout<<"\n\tMax element in tree is= "<<pre->data;

}

node\* binary\_tree::search (int target)

{

node \*found,\*pre,\*temp; found=NULL; if(root!=NULL)

{

temp=root; while(temp!=NULL)

{

if (temp->data>target)

{

pre=temp; temp=temp->left;

}

else if(temp->data<target)

{

}

}

else

}

else

{

}

pre=temp; temp=temp->right;

found=temp; break;

cout<<"\n\t Tree is Empty..."; return found;

}

void binary\_tree::deletenode (int target)

{

node \*location,\*parent; parent = NULL ;

location = search(target); if (location != NULL )

if ( location -> left != NULL && location -> right != NULL )

{

node\* xsucc; parent = location ;

xsucc = location -> right ; while ( xsucc -> left != NULL )

{

parent = xsucc ; xsucc = xsucc -> left ;

}

location -> data = xsucc -> data ; location = xsucc ;

}

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

{

if ( parent -> right == location )

parent -> right = NULL ;

else

parent -> left = NULL ;

delete location;

}

if ( location -> left == NULL && location -> right != NULL )

{

if ( parent -> left == location )

parent -> left = location -> right ;

else

parent -> right = location -> right ;

delete location;

}

if ( location -> left != NULL && location -> right == NULL )

{

if ( parent -> left == location )

parent -> left = location -> left ;

else

parent -> right = location -> left ;

}

}

else

delete location;

}

int main()

cout<<"\n\t Data which is to be deleted, not found";

{

binary\_tree b; node \*\*p;

int count, data, target; char choice,option; do

{

cout<<"\n\t Select Operation for Binary Search Tree \n"; cout<<"\n\t1. To Binary Tree Creation";

cout<<"\n\t2. To Display Binary Search Tree (Inorder) Acending Sorting"; cout<<"\n\t3. To Display Binary Search Tree (Preorder)";

cout<<"\n\t4. To Display Binary Search Tree (Postorder)"; cout<<"\n\t5. To Display Binary Search Tree Decending Sorting"; cout<<"\n\t6. To Delete node from Binary Search Tree"; cout<<"\n\t7. To find max element in the Binary Search Tree"; cout<<"\n\t8. To find min element in the Binary Search Tree"; cout<<"\n\t9. To Search element in the Binary Search Tree"; cout<<"\n\t0. To Exit";

cout<<"\n\n\t Enter your choice:- "; cin>>option;

switch(option)

{

case '1':

{

cout<<"\n \t How many node you want to added in the Binary Search Tree: "; cin>>count;

for(int i=0;i<count;i++)

{

node: ";

}

break;

}

cout<<"\n \t Enter data which is need to be added in a Binary Search Tree

cin>>data; b.insert (data);

case '2':

{

}

case '3':

{

}

case '4':

{

}

case '5':

{

}

case '6':

{

cout<<"\n\tBinary Search Tree (Inorder)"; b.inorder(b.root);

break;

cout<<"\n\tBinary Search Tree (Preorder)"; b.preorder(b.root);

break;

cout<<"\n\tBinary Search Tree (Postorder)"; b.postorder(b.root);

break;

cout<<"\n\tBinary Search Tree (Sorted)"; b.rightorder(b.root);

break;

";

}

case '7':

{

}

case '8':

{

}

case '9':

{

cout<<"\n\t Enter the element which is to be deleted from the Binary Search Tree:

cin>>target; b.deletenode(target); break;

b.maxelement(); break;

b.minelement(); break;

cout<<"\n\t Enter the element which is to be search in the Binary Search Tree: "; cin>>target;

if(b.search(target))

cout<<"\n\t Element is found in the Binary Search Tree";

}

case '0':

{

}

else break;

break;

cout<<"\n\t Element is not found in the Binary Search Tree";

default:

{

}

}

cout<<"\n \tInvalid Option"; break;

cout<<"\n \tDo you want to continue:- "; cin>>choice;

}while(choice!='n');

}