Aizaz Sharif

P12-6006

BS(CS)

#include<iostream>

#include<conio.h>

#include<string>

using namespace std;

bool found=false;

struct node

{

int info;

node \*left;

node \*right;

};

class BST

{

private:

node \*tem;

node \*temp;

node \*supporting;

node \*immediate;

int x;

int y;

node \*temp2;

node \*temp1;

node \*temp3;

public:

int \*current;

int height;

int n;

node \*root;

BST()

{

temp=root;

temp=root=NULL;

}

void Insert(node \*temp,node \*temp1)

{

if(root==NULL)

{

root=new node;

root->info=n;

root->left=NULL;

root->right=NULL;

return;

}

if(temp->info==n)

{

cout<<"Its already there"<<endl;

return;

}

if(temp->info>n)

{

if(temp->left!=NULL)

{

Insert(temp->left,temp);

}

else

{

temp->left=new node;

temp->left->info=n;

temp->left->left=NULL;

temp->left->right=NULL;

cout<<"Its parent is "<<temp1->info<<endl;

/\*return;\*/

}

}

if(temp->info<n)

{

if(temp->right!=NULL)

{

Insert(temp->right,temp);

}

else

{

temp->right=new node;

temp->right->info=n;

temp->right->left=NULL;

temp->right->right=NULL;

cout<<"Its parent is "<<temp1->info<<endl;

/\*return;\*/

}

}

x=BF(temp);

if(x==2)

{

if(temp==root)

{case1And2(temp);}

else

{

if(temp->left->info>n && temp1->info<temp->info)//case1

{

temp2=temp1->right->left;

temp1->right->left=temp2->right;

temp2->right=temp;

temp1->right=temp2;

temp=temp2;

cout<<"case 1 "<<endl;

return;

}

if(temp->left->info>n && temp1->info>temp->info)//case1

{

temp2=temp1->left->left;

temp1->left->left=temp2->right;

temp2->right=temp;

temp1->left=temp2;

temp=temp2;

cout<<"case 1 "<<endl;

return;

}

if(temp->left->info<n && temp1->info<temp->info)//case2

{

temp2=temp1->right->left;

temp3=temp2->right;

temp2->right=temp3->left;

temp3->left=temp2;

temp1->right->left=temp3->right;

temp3->right=temp;

temp1->right=temp3;

temp=temp2;

cout<<"case 2 "<<endl;

return;

}

if(temp->left->info<n && temp1->info>temp->info)//case2

{

temp2=temp1->left->left;

temp3=temp2->right;

temp2->right=temp3->left;

temp3->left=temp2;

temp1->left->left=temp3->right;

temp3->right=temp;

temp1->left=temp3;

temp=temp2;

cout<<"case 2 "<<endl;

return;

}

}

}

if(x==-2)

{

if(temp==root)

{case3And4(temp);}

else

{

if(temp->right->info<n && temp1->info<temp->info)//case4

{

temp2=temp1->right->right;

temp1->right->right=temp2->left;

temp2->left=temp;

temp1->right=temp2;

temp=temp2;

cout<<"case 4 "<<endl;

return;

}

if(temp->right->info<n && temp1->info>temp->info)//case4

{

temp2=temp1->left->right;

temp1->left->right=temp2->left;

temp2->left=temp;

temp1->left=temp2;

temp=temp2;

cout<<"case 4 "<<endl;

return;

}

if(temp->right->info>n && temp1->info<temp->info)//case3

{

temp2=temp1->right->right;

temp3=temp2->left;

temp2->left=temp3->right;

temp3->right=temp2;

temp1->right->right=temp3->left;

temp3->left=temp;

temp1->right=temp3;

temp=temp2;

cout<<"case 3 "<<endl;

return;

}

if(temp->right->info>n && temp1->info>temp->info)//case3

{

temp2=temp1->left->right;

temp3=temp2->left;

temp2->left=temp3->right;

temp3->right=temp2;

temp1->right->right=temp3->left;

temp3->left=temp;

temp1->left=temp3;

temp=temp2;

cout<<"case 3 "<<endl;

return;

}

}

}

return;

}

//---------------------------------------------------//

//---------------------------------------------------//

int BF(node \*temp)

{

int leftH=0;

int rightH=0;

if(temp->left!=NULL)

{

height++;

HeightFunction(temp->left);

leftH=height;

}

height=0;

if(temp->right!=NULL)

{

height++;

HeightFunction(temp->right);

}

else

{

height=0;

}

rightH=leftH-height;

height=0;

cout<<" the balance factor of "<<temp->info<<" is "<<rightH<<endl;

return rightH;

}

//---------------------------------------------------//

void HeightFunction(node\*temp)

{

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

return;

if(temp->left!=NULL)

{

height++;

HeightFunction(temp->left);

}

if(temp->right!=NULL)

{

height++;

HeightFunction(temp->right);

}

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

{

height--;

}

}

//---------------------------------------------//

//---------------------------------------------//

void case3And4(node \*temp)

{

if(temp->right->info>n)//case 3

{

temp2=temp->right;//right roation

temp3=temp2->left;

temp2->left=temp3->right;

temp3->right=temp2;

temp->right=temp3->left;//left rotation

temp3->left=temp;

root=temp3;

cout<<"case 3"<<endl;

return;

}

else if(temp->right->info<n)//case4

{

temp2=temp->right;

temp->right=temp2->left;

temp2->left=temp;

root=temp2;

cout<<"case 4"<<endl;

return;

}

}

void case1And2(node \*temp)

{

if(temp->left->info>n)//case1

{

temp2=temp->left;

temp->left=temp2->right;

temp2->right=temp;

root=temp2;

cout<<"case 1"<<endl;

return;

}

else if(temp->left->info<n)//case2

{

temp2=temp->left;//left rotation

temp3=temp2->right;

temp2->right=temp3->left;

temp3->left=temp2;

temp->left=temp3->right;//right rotation

temp3->right=temp;

root=temp3;

cout<<"case 2"<<endl;

return;

}

}

//--------------------------------------------//

//--------------------------------------------//

void inOrder(node \*temp)

{

if(root==NULL)

{

cout<<" Tree is empty. And so is my brain "<<endl;

return;

}

if( temp->left!=NULL )

inOrder(temp->left);

cout<<temp->info<<" ";

if( temp->right!=NULL )

inOrder(temp->right);

/\*cout<<temp->right<<" ";\*/

return;

}

//----------------------------------------------------//

//----------------------------------------------------//

void Delete (node \*temp,node\*temp1)

{

if( root==NULL )

{

cout<<" Tree is NULL already ";

return;

}

if( temp->info==n)

{

if(temp->left==NULL && temp->right==NULL)//<<<<<<<-- When the deleting node is a leaf //

{

if(temp==root)

root=NULL;

delete temp;

if(temp2->left==temp)

temp2->left=NULL;

else

temp2->right=NULL;

}

if(temp->left!=NULL && temp->right!=NULL)//<<<<<<<-- When the deleting node has child on both sides--------//

{

immediate=temp->right;

if(immediate->left!=NULL)

{

while(immediate->left!=NULL)

{

supporting=immediate;

immediate=immediate->left;

}

temp->info=immediate->info;

supporting->left=NULL;

delete immediate;

immediate=NULL;

}

else

{

temp->info=immediate->info;

temp->right=immediate->right;

delete immediate;

immediate=NULL;

}

return;

}

if(temp->left!=NULL && temp->right==NULL)//<<<<<<<<-- When the deleting node has only left child----//

{

if(temp==root)

{

delete root;

root =temp->left;

}

else

{

temp2->left=temp->left;

delete temp;

}

return;

}

if(temp->right!=NULL && temp->left==NULL)//<<<<<<<<-- When the deleting node has only right child----//

{

if(temp==root)

{

delete root;

root =temp->right;

}

else

{

temp2->right=temp->right;

delete temp;

}

return;

}

//>>>>>>---when the present temp is not the required one to delete<<<<<<<--//

else if(temp->info > n)

{

temp2=temp->right;

Delete(temp->left,temp);

}

else if(temp->info < n)

{

temp2=temp->left;

Delete(temp->right,temp);

}

else

{

;

}

}

}

~BST()

{

delete root;

}

};

void main()

{

int x=0;

/\*int n=0;\*/

BST A;//Insertion

do{

cout<<"Want to add a number in tree? Press 1"<<endl;

cin>>x;

if(x==1)

{

cout<<"Which number you want to send?"<<endl;

cin>>A.n;

A.Insert(A.root,A.root);

/\*A.calculate(A.root);\*/

}

}while(x==1);

A.inOrder(A.root);//Print

do{//Deletion

cout<<"Want to Delete a number in tree? Press 2"<<endl;

cin>>x;

if(x==2)

{

cout<<"Which number you want to delete?"<<endl;

cin>>A.n;

A.Delete(A.root,A.root);

A.inOrder(A.root);

}

}while(x==2);

A.inOrder(A.root);

getch();

}