Assignment: Data Structures

Roll no :20p-0480

Name:Muhammad Shaheer

Problem: 1 Construct an empty BST.

◼ Determine if the BST is empty.

◼ Search the BST for a given item.

◼ Insert a new item in the BST and maintain the

BST property.

◼ Delete the item from BST and maintain the BST

property.

◼ Traverse the BST visiting each node exactly once

using any traversal method.

Code:

#include<iostream>

using namespace std;

    class node

    {

    public:

    int data;

    node\* left;

    node\* right;

    node(int data)

    {

        this->data=  data;

        left= NULL;

        right= NULL;

    }

    };

class BinarySearchTree

{

    private:

    //node class

    //private data members and methods

    node\* root;

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

    node\* Delete(node\* root,int data);

    node\* InOrderTraversal( node\* root);

    node\* searchNode(node\* root,int key);

    node\* FindMax(node\* root);

    bool isEmpty(node\* root);

    public:

    BinarySearchTree()

    {

        root= NULL;

    }

    void    Insert(int val)

    {

       Insert(this->root,  val);

    }

    bool    isEmpty()

    {

       return isEmpty(root);

    }

    void    Delete(int val)

    {

   Delete(this->root, val);

    }

    void InOrderTraversal()

    {

        InOrderTraversal( this->root);

    }

    void searchNode(int key)

    {

        node\* result = searchNode(root, key);

        if (result == NULL)

        {

            cout << "Node " << key << " not found. Enter valid node" << endl;

        } else

        {

            cout << "Node  " << key << " found." << endl;

        }

    }

};

node\* BinarySearchTree::Insert(node\* r, int val )

{

 if (r==NULL)

    {

        node\* t= new node(val);

        if (r==root)

            root=t;

        else

        r=t;

    }

    else if (r->data== val)

    {

        cout<<"Record already exist"<<val;

    }

    else if (val < r->data) // insert on left s

    r->left = Insert(r->left , val );

    else if (val > r->data)

        r->right= Insert( r->right,val);

    return r;

}

node \* BinarySearchTree::Delete(node\* r, int data)

{

    if(r==NULL)

     return r;

    else if(data < r->data)

        r->left = Delete(r->left, data);

    else if (data> r->data)

        r->right = Delete(r->right, data);

    else

    {

        //No child

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

        {

            delete r;

            r = NULL;

            return r;

        }

        //One child on left

        else if(r->right == NULL)

        {

            node\* temp = r;

            r= r->left;

            delete temp;

        }

        //One child on right

        else if(r->left == NULL)

        {

            node\* temp = r;

            r= r->right;

            delete temp;

        }

        //two child

        else

        {

            node\* temp = FindMax(root->left);

            root->data = temp->data;

            r->left = Delete(root->left, temp->data);

        }

    }

    return r;

}

node \* BinarySearchTree::InOrderTraversal( node\* r)

{

     if (r == NULL)

        return NULL;

    //first recur on left child

    InOrderTraversal(r->left);

   //then print the data of node

    cout << " "<< r->data << " -> ";

  // now recur on right child

    InOrderTraversal(r->right);

}

node\* BinarySearchTree::FindMax(node\* r)

{

    while(r->right!=NULL){

        r= r->right;

    }

    return r;

}

node\* BinarySearchTree:: searchNode(node \*r, int key) {

            if (r == nullptr || r->data == key)

            {

                return r;

            }

            else if (key < r->data)

            {

                return searchNode(r->left, key);

            }

             else

            {

                return searchNode(r->right, key);

            }

        }

bool BinarySearchTree::isEmpty(node\* root)

{

    if (root == NULL)

    {

        cout<<"BST is empty"<<endl;

    }

    else

    {

        cout<<"BST is not empty"<<endl;

    }

}

int main (){

    BinarySearchTree BST;

    cout<<"To check emptiness of BST"<<endl;

    BST.isEmpty();

    BST.Insert(10);

    BST.Insert(8);

    BST.Insert(6);

    BST.Insert(9);

    BST.Insert(15);

    BST.Insert(14);

    BST.Insert(20);

    cout<<endl;

    cout<<"Insertion in BST in inorder Traversal Method :"<<endl;

    BST.InOrderTraversal();

    cout<<endl;

    cout<<"After deleting node:"<<endl;

    BST.Delete(10);

    BST.InOrderTraversal();

    cout<<endl;

    cout<<"Searching node in BST:"<<endl;

    BST.searchNode(15);

    cout<<endl;

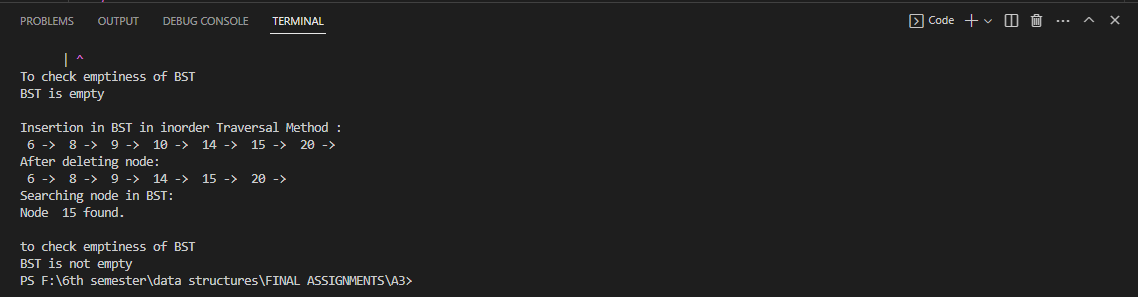
    cout<<"to check emptiness of BST"<<endl;

    BST.isEmpty();

    return 0;

}

Output:



The END------------------------------------------------------------------------------------------------------------------------------