Task-5

#include<iostream>

#include<math.h>

#include<stack>

using namespace std;

//creating a struct for tree node

struct Tree{

    Tree \*left,\*right;

    int data;

    Tree(int d){

        left = NULL;

        right = NULL;

        data = d;

    }

};

//craeting a  Binary Tree ADT

class BT{

    Tree \*root;

public:

//Constructor

    BT(Tree \*r = NULL){

        root = r;

    }

    //Method to insert

    void insert(int data){

        Tree \*newNode = new Tree(data);

        //if tree does't exist craete a root

        if(root==NULL){

            root = newNode;

            return;

        }

        Tree \* temp = root;

        //Finding the place of inseering according to rules of BST

        while (temp!=NULL){

            //if data less than root move to left

            if(temp->data>data){

                if(temp->left)

                temp = temp->left;

            else {

               temp->left =  newNode;

               break;

            }

            }

            //if data greate than root move to right

            else if(temp->data<data){

                if(temp->right){

                        temp = temp->right;

                }

                    else{

                        temp->right = newNode;

                       break;

                    }

                }

            //if duplicate found don't insert

            else{

                cout<<"Duplicate Value\n";

                               break;

            }

       }

    }

    // to search any node

    Tree\* search(int data){

        //calling a finding node method

        return find(root,data);

    }

    //Method to display

    void pre\_order\_interative(){

        pre\_order(root);

        cout<<endl;

    }

    void post\_order\_interative() {

        post\_order(root);

        cout << endl;

    }

    int getmax(){

        return max(root)->data;

    }

    int getmin(){

        return min(root)->data;

    }

    Tree \* GetRoot(){

        return root;

    }

    void deleteNode(int key){

         del(root,key);

    }

    int isBST(Tree \*n)

    {

    if (n == NULL)

        return 1;

    if (n->left != NULL && max(n->left)->data > n->data) // it will be false if the max of left is greater

        return 0;

    if (n->right != NULL && min(n->right)->data < n->data) // it will be false if the min of right is less or equal

        return 0;

    if (!isBST(n->left) || !isBST(n->right))

        return 0;

    return 1;

}

private:

//recursily find the desired node if not found simpple return

    Tree\* find(Tree\*n,int data){

         if(n == NULL)

        return n;

        else if(n->data == data)

        return n;

        else if(n->data<data){

         return find(n->right,data);

        }

        else if(n->data>data){

         return find(n->left,data);

        }

    }

    void post\_order(Tree\* n) {

        {

            if (root == NULL)

                return;

            stack<Tree\*> st1;

            st1.push(n);

            stack<int> st2;

            Tree\* temp = n;

            while (!st1.empty()){

                st2.push(temp->data);

            if (temp->left)

                st1.push(temp->left);

            if (temp->right)

                st1.push(temp->right);

        }

        while (!st2.empty()) {

              cout << st2.top();

              st2.pop();

            }

        }

    }

    void pre\_order(Tree \*n) {

            if (n == NULL)

                return;

            stack<Tree\*> st;

            st.push(n);

            //Keep pushing and printing until stack is not empty

            while (st.empty() == false)

            {

                Tree\* node = st.top();

                //first root printed

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

                st.pop();

                //right inserted first cuz of FIFO

                if (node->right)

                    st.push(node->right);

                if (node->left)

                    st.push(node->left);

            }

    }

    Tree\* min(Tree\* n) {

        // to find the max

        if (n==NULL)

            return NULL;

        else if (n->left) {

            return min(n -> left);

        }

        else

            return n;

    }

    Tree\* max(Tree\* n) {

        // to find the max

        if (n==NULL)

            return NULL;

        else if (n->right) {

            return max(n -> right);

        }

        else

            return n;

    }

    Tree\* del(Tree\* r, int key)

    {

        if (r == NULL)

            return r;

        if (key < r->data)

            r->left = del(r->left, key);

        else if (key > r->data)

            r->right = del(r->right, key);

        else {

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

                return NULL;

            else if (r->left == NULL) {

                Tree\* temp = r->right;

                delete r;

                r = NULL;

                return temp;

            }

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

                Tree\* temp = r->left;

                delete r;

                r = NULL;

                return temp;

            }

            Tree\* temp = min(r->right);

            r->data = temp->data;

            r->right = del(r->right, temp->data);

        }

        return r;

    }

};

int main(){

    BT t;

    bool exit = false;

    int choice;

    Tree\* n;

    int data;

    while (!exit)  {

        cout<<"1-Insert\n2-Search\n3-Display Preorder Interative\n4-Display PostOrder\n5-IsBST\n0-Eixt\nChosse option: ";

        cin>>choice;

        switch (choice)

        {

        case 1:

        cout<<"Enter Data: ";

        cin>>data;

        t.insert(data);

            break;

        case 2:

        cout<<"Enter Data to search: ";

        cin>>data;

        cout<<t.search(data)->data<<endl;

        break;

        case 3:

         t.pre\_order\_interative();

        break;

        case 4:

        t.post\_order\_interative();

        break;

        case 5:

       cout<<"isBST: " <<t.isBST(t.GetRoot())<<endl;

        case 0:

        exit = true;

          break;

        }

        system("pause");

        system("cls");

    }

}

