DATA STRUCTURES AND ALGORITHMS

ASSIGNMENT # 4

Hassaan Akbar Cheema

Regn no. # 174351

Instructor : Sir Abid Rauf

//Binary Search Tree Program

#include <iostream>

#include <cstdlib>

using namespace std;

int i=0;

// binary search class

class BinarySearchTree

{

private: // private structure

struct tree\_node

{

tree\_node\* left;

tree\_node\* right;

int data;

};

tree\_node\* root; // pointer to store root address

public:

BinarySearchTree()//method to set root null

{

root = NULL;

}

bool isEmpty() const { return root==NULL; } //method return true if tree is empty

void print\_inorder(); // method to print in-order form

void inorder(tree\_node\*);

void print\_preorder(); // method to print pre-order form

void preorder(tree\_node\*);

void print\_postorder(); // method to print post-order form

void postorder(tree\_node\*);

void insert(int);

};

// Smaller elements go left

// larger elements go right

void BinarySearchTree::insert(int d)

{

tree\_node\* t = new tree\_node; // creating new node

tree\_node\* parent; // alternative tree node type pointer to store parent node address

t->data = d; // feeding value in data part of new node

t->left = NULL; // no node at left so address is null

t->right = NULL; // no node at right so address is null

parent = NULL; // pointing no parent node yet

int flag = 0;

// is this a new tree?

if(isEmpty()) root = t;

else

{

tree\_node\* curr; // alternative tree node type pointer to store current node address

curr = root; // starting from root

// Find the Node's parent

while(curr) // while current is not null

{

parent = curr; // current is parent

if(t->data > curr->data && t-> data != curr-> data) curr = curr->right; // if data of new node is greater than parent

// go right else go left

else if (t->data < curr->data && t-> data != curr-> data) curr = curr->left;

else{

cout << "duplicate found"<<endl;

--i;

return;

}

}

if(t->data < parent->data)

parent->left = t;// when last node is found enter the address of new node to either right or left

// of the current/parent node

else

parent->right = t;

}

}

void BinarySearchTree::print\_inorder()

{

inorder(root); // call in-order method

}

void BinarySearchTree::inorder(tree\_node\* p)

{

if(p != NULL) // if parent node is not null

{

if(p->left) inorder(p->left); // if left address of parent node is not null than

// call the same function recursively to reach the last node

cout<<" "<<p->data<<" ";// when the last node is found print the data

if(p->right) inorder(p->right);// if right address of parent node is not null than

// call the same function recursively to reach the last node

}

else return;

}

void BinarySearchTree::print\_preorder()

{

preorder(root); // call pre-order method

}

void BinarySearchTree::preorder(tree\_node\* p)

{

if(p != NULL) // if parent node is not null

{

cout<<" "<<p->data<<" "; // when the last node is found print the data

if(p->left) preorder(p->left); // if left address of parent node is not null than

// call the same function recursively to reach the last node

if(p->right) preorder(p->right);// if right address of parent node is not null than

// call the same function recursively to reach the last node

}

else return;

}

void BinarySearchTree::print\_postorder()

{

postorder(root);// call post-order method

}

void BinarySearchTree::postorder(tree\_node\* p)

{

if(p != NULL)

{

if(p->left) postorder(p->left);// if left address of parent node is not null than

// call the same function recursively to reach the last node

if(p->right) postorder(p->right);// if right address of parent node is not null than

// call the same function recursively to reach the last node

cout<<" "<<p->data<<" ";// when the last node is found print the data

}

else return;

}

int main()

{

BinarySearchTree b;

int choice,tmp,tmp1;

int num;

cout<<" Enter how many values to be inserted : ";

cin>>num;

for (i; i < num; i++){

cout<<" Enter Number to be inserted : ";

cin>>tmp;

b.insert(tmp);

}

cout<<" INORDER : ";

b.print\_inorder();

cout<< endl;

cout<<" PRE\_ORDER : ";

b.print\_preorder();

cout<< endl;

cout<<" POST\_ORDER : ";

b.print\_postorder();

cout<< endl;

return 0;

}

OUTPUT:

