**EXPERIMENT-14**

**AIM :** Create a Binary Tree and perform Tree traversals (Preorder, Inorder, Postorder) using the concept of recursion.

**ALGORITHM :**

**SOURCE CODE :**

#include <bits/stdc++.h>

using namespace std;

class BinaryTree {

public :

class Node {

public :

int data;

Node\* left;

Node\* right;

Node(int data) {

this->data=data;

}

};

Node\* root;

BinaryTree(int level[]) {

root=new Node(level[0]);

queue<Node\*> q;

q.push(root);

int idx=1;

while (!q.empty()) {

Node\* store=q.front();

q.pop();

if (level[idx]!=-1) {

store->left=new Node(level[idx]);

q.push(store->left);

}

idx++;

if (level[idx]!=-1) {

store->right=new Node(level[idx]);

q.push(store->right);

}

idx++;

}

}

void preorder() {

preorder(root);

cout << endl;

}

void preorder(Node\* node) {

if (node==NULL) {

return;

}

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

preorder(node->left);

preorder(node->right);

}

void inorder() {

inorder(root);

cout << endl;

}

void inorder(Node\* node) {

if (node==NULL) {

return;

}

inorder(node->left);

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

inorder(node->right);

}

void postorder() {

postorder(root);

cout << endl;

}

void postorder(Node\* node) {

if (node==NULL) {

return;

}

postorder(node->left);

postorder(node->right);

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

}

};

int main() {

int level[25]={70,24,50,17,-1,60,24,-1,19,-1,-1,84,-1,-1,18,-1,90,15,-1,-1,93,-1,-1,-1,-1};

cout << "Level order traversal of binary tree : " << endl;

for (int idx=0;idx<25;idx++) {

cout << level[idx] << " ";

}

cout << endl;

BinaryTree BT(level);

cout << "Binary tree created !!!" << endl;

cout << "Preorder traversal : " << endl;

BT.preorder();

cout << "Inorder traversal : " << endl;

BT.inorder();

cout << "Postorder traversal : " << endl;

BT.postorder();

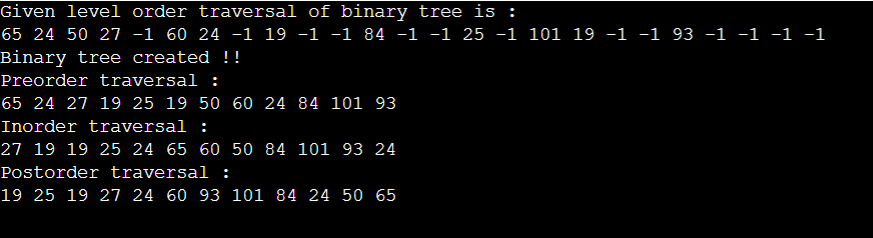
return 0;

}

**EXPERIMENT-14**

**AIM :** Create a Binary Tree and perform Tree traversals (Preorder, Inorder , Postorder) using the concept of recursion.

**OUTPUT:**



**EXPERIMENT-15**

**AIM:** Create a Binary Search Tree and perform insertion and deletion.

**ALGORITHM:**

**SOURCE CODE:**

#include <bits/stdc++.h>

using namespace std;

class BinarySearchTree {

public :

class Node {

public :

int data;

Node\* left;

Node\* right;

Node(int data) {

this->data=data;

}

};

Node\* root=NULL;

void insert(int value) {

root=insert(root,value);

cout << "Inorder traversal of BST is : " << endl;

inorder(root);

cout << endl;

}

Node\* insert(Node\* node,int value) {

if (node==NULL) {

return new Node(value);

}

if (node->data<value) {

node->right=insert(node->right,value);

}

else {

node->left=insert(node->left,value);

}

return node;

}

void toDelete(int value) {

root=toDelete(root,value);

cout << "Inorder traversal of BST is : " << endl;

inorder(root);

cout << endl;

}

Node\* toDelete(Node\* node,int value) {

if (node->data<value) {

node->right=toDelete(node->right,value);

}

else if (node->data>value) {

node->left=toDelete(node->left,value);

}

else {

if (node->left==NULL) {

return node->right;

}

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

return node->left;

}

else {

node->data=max(node->left);

node->left=toDelete(node->left,node->data);

}

}

return node;

}

int max(Node\* node) {

if (node->right==NULL) {

return node->data;

}

return max(node->right);

}

void inorder(Node\* node) {

if (node==NULL) {

return;

}

inorder(node->left);

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

inorder(node->right);

}

};

int main() {

BinarySearchTree BST;

int a;

while (true){

cout << "1.Inserting 2.Delete" << endl;

cin>>a;

if (a==1){

int b;

cout<<"Enter Value: ";

cin>>b;

BST.insert(b);

}

else if(a==2){

int b;

cout<<"Enter Value: ";

cin>>b;

BST.toDelete(b);

}

}

return 0;

}

**EXPERIMENT-15**

**AIM:** Create a Binary Search Tree and perform insertion and deletion.

**OUTPUT:**

