**/\* Deleting a node from Binary search tree \*/**

#include<iostream>

using namespace std;

struct Node {

int data;

struct Node \*left;

struct Node \*right;

};

**//Function to find minimum in a tree.**

Node\* FindMin(Node\* root)

{

while(root->left != NULL)

root = root->left;

return root;

}

**// Function to search a delete a value from tree.**

struct Node\* Delete(struct Node \*root, int data) {

if(root == NULL)

return root;

else if(data < root->data)

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

else if (data > root->data)

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

// Wohoo... I found you, Get ready to be deleted

else {

// Case 1: No child

if(root->left == NULL && root->right == NULL) {

delete root;

root = NULL;

}

//Case 2: One child

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

struct Node \*temp = root;

root = root->right;

delete temp;

}

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

struct Node \*temp = root;

root = root->left;

delete temp;

}

// case 3: 2 children

else {

struct Node \*temp = FindMin(root->right);

root->data = temp->data;

root->right = Delete(root->right,temp->data);

}

}

return root;

}

**//Function to visit nodes in Inorder**

void Inorder(Node \*root) {

if(root == NULL)

return;

Inorder(root->left); //Visit left subtree

printf("%d ",root->data); //Print data

Inorder(root->right); // Visit right subtree

}

***// Function to Insert Node in a Binary Search Tree***

Node\* Insert(Node \*root,char data) {

if(root == NULL) {

root = new Node();

root->data = data;

root->left = root->right = NULL;

}

else if(data <= root->data)

root->left = Insert(root->left,data);

else

root->right = Insert(root->right,data);

return root;

}

int main() {

***/\*Code To Test the logic***

Creating an example tree

5

/ \

3 10

/ \ \

1 4 11

\*/

Node\* root = NULL;

root = Insert(root,5);

root = Insert(root,10);

root = Insert(root,3);

root = Insert(root,4);

root = Insert(root,1);

root = Insert(root,11);

***// Deleting node with value 5, change this value to test other cases***

root = Delete(root,5);

//Print Nodes in Inorder

cout<<"Inorder: ";

Inorder(root);

cout<<"\n";

}