|  |  |
| --- | --- |
| Name | Taaha Hussain Khan |
| Roll No | L1F21BSCS0917 |
| Section | D12 |
|  |  |

**TASK 01**

#include <iostream>

using namespace std;

template <typename T>

struct Node {

T data;

Node\* left;

Node\* right;

};

template <typename T>

class BST {

public:

Node<T>\* root;

BST() {

root = nullptr;

}

void insertRec(Node<T>\*& p, Node<T>\* newNode) {

if (p == nullptr) {

p = newNode;

return;

}

if (newNode->data < p->data) {

insertRec(p->left, newNode);

}

else {

insertRec(p->right, newNode);

}

}

void postorder(Node<T>\* p) {

if (p == nullptr) {

return;

}

postorder(p->left);

postorder(p->right);

cout << p->data << " ";

}

void INSERT(T d) {

Node<T>\* newNode = new Node<T>{ d, NULL, NULL };

insertRec(root, newNode);

}

};

int main() {

BST<int> tree;

tree.INSERT(100);

tree.INSERT(50);

tree.INSERT(75);

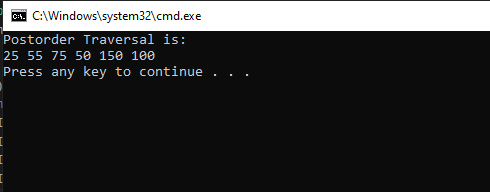
tree.INSERT(150);

cout << "Postorder Traversal is: " << endl;

tree.postorder(tree.root);

return 0;

}



**Task02**

#include <iostream>

using namespace std;

struct node {

int key;

struct node \*left, \*right;

};

struct node\* newNode(int item)

{

struct node\* temp

= (struct node\*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node\* root)

{

if (root != NULL) {

inorder(root->left);

cout << root->key << " ";

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

if (node == NULL)

return newNode(key);

if (key < node->key)

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

else

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

return node;

}

struct node\* minValueNode(struct node\* node)

{

struct node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key)

{

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

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

return NULL;

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

struct node\* temp = root->right;

free(root);

return temp;

}

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

struct node\* temp = root->left;

free(root);

return temp;

}

struct node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main()

{

struct node\* root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

cout << "Inorder traversal of the tree \n";

inorder(root);

cout << endl << endl;

cout << "\n\t\tTEST CASES" << endl;

//1. Delete a value that is not present in a BST

cout << "===> Deleting a value that is not present in a BST " << endl;

root = deleteNode(root, 200);

cout << "Inorder traversal of the modified tree \n";

cout << endl;

inorder(root);

cout << endl;

// 2. Delete the value present at the root in such a way that:

// 3. Root has no children

cout << "===> Root has no children" << endl;

root = deleteNode(root, 50);

cout << "Inorder traversal of the modified tree \n";

cout << endl;

inorder(root);

cout << endl;

// 4. Root has only one leaf child

cout << "===>Root has only one leaf child" << endl;

root = insert(root, 50);

root = deleteNode(root, 30);

cout << "Inorder traversal of the modified tree \n";

cout << endl;

inorder(root);

cout << endl;

// 5. Root has only one NON-LEAF child

cout << "===>Root has only one NON-LEAF child" << endl;

root = insert(root, 30);

root = deleteNode(root, 20);

cout << "Inorder traversal of the modified tree \n";

cout << endl;

inorder(root);

cout << endl;

// 6. Root has two leaf children

cout << "===> Root has two leaf children. " << endl;

root = insert(root, 20);

root = deleteNode(root, 40);

cout << "Inorder traversal of the modified tree \n";

cout << endl;

inorder(root);

cout << endl;

// 7. Root had two NON-LEAF children

cout << "===> Root had two NON-LEAF children" << endl;

root = insert(root, 40);

root = deleteNode(root, 60);

cout << "Inorder traversal of the modified tree \n";

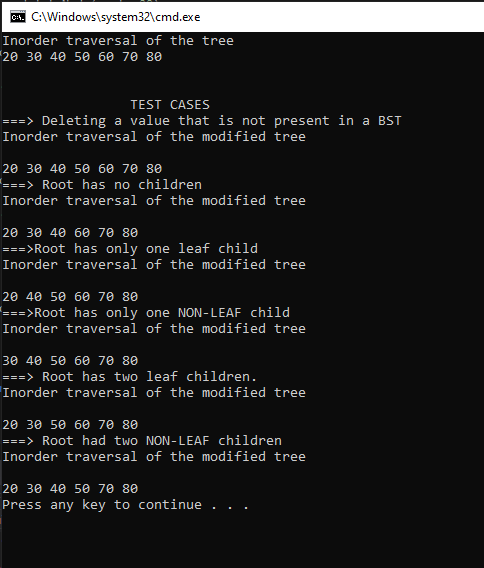
cout << endl;

inorder(root);

cout << endl;

return 0;

}



**Task 03**

#include <iostream>

using namespace std;

class BST {

int data;

BST\* left;

BST\* right;

public:

BST()

{

data = 0;

right = NULL;

left = NULL;

}

// constructor

BST(int d)

{

data = d;

right = NULL;

left = NULL;

}

BST\* Insert(BST\* root, int d)

{

if (root == NULL)

{

return new BST(d);

}

if (d > root->data)

{

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

}

else if (d < root->data)

{

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

}

return root;

}

void Inorder(BST\* root)

{

if (!root)

{

return;

}

Inorder(root->left);

cout << root->data << ", ";

Inorder(root->right);

}

void Preorder(BST\* root)

{

if (!root)

{

return;

}

cout << root->data << ", ";

Preorder(root->left);

Preorder(root->right);

}

void Postorder(BST\* root)

{

if (!root)

{

return;

}

Postorder(root->left);

Postorder(root->right);

cout << root->data << ", ";

}

bool search(BST\* root, int key)

{

if (root == NULL)

{

return false;

}

if (root->data == key)

{

return true;

}

if (key > root->data)

{

return search(root->right, key);

}

else

{

return search(root->left, key);

}

return false;

}

~BST()

{

if (left != NULL)

delete left;

if (right != NULL)

delete right;

}

};

int main()

{

BST b, \*test = NULL;

test = b.Insert(test, 50);

b.Insert(test, 30);

b.Insert(test, 20);

b.Insert(test, 40);

b.Insert(test, 70);

b.Insert(test, 60);

b.Insert(test, 80);

cout << "\n\nIn order traversal: \n";

b.Inorder(test);

cout << "\n\nPre order traversal: \n";

b.Preorder(test);

cout << "\n\nPost order traversal: \n";

b.Postorder(test);

delete test;

cout << "\n\nAfter deletion : " << endl;

cout << "\n\nIn order traversal: \n";

b.Inorder(test);

cout << "\n\nPre order traversal: \n";

b.Preorder(test);

cout << "\n\nPost order traversal: \n";

b.Postorder(test);

return 0;

}

