**NAME:SAYALI JIVAN CHAUDHARI**

**ROLL NO.:14**

**PRN NO.2023015400005055**

**1)implementation of BST OPERATIONS(insert delete and display)**

// C++ program to implement optimized delete,insert and display in BST.

#include <bits/stdc++>

#include<iostream>

using namespace std;

struct Node {

int key;

struct Node \*left, \*right;

};

// A utility function to create a new BST node

Node\* newNode(int item)

{

Node\* temp = new Node;

temp->key = item;

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

return temp;

}

// A utility function to do inorder traversal of BST

void inorder(Node\* root)

{

if (root != NULL) {

inorder(root->left);

printf("%d ", root->key);

inorder(root->right);

}

}

/\* A utility function to insert a new node with given key in

\* BST \*/

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

{

/\* If the tree is empty, return a new node \*/

if (node == NULL)

return newNode(key);

/\* Otherwise, recur down the tree \*/

if (key < node->key)

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

else

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

/\* return the (unchanged) node pointer \*/

return node;

}

/\* Given a binary search tree and a key, this function

deletes the key and returns the new root \*/

Node\* deleteNode(Node\* root, int k)

{

// Base case

if (root == NULL)

return root;

// Recursive calls for ancestors of

// node to be deleted

if (root->key > k) {

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

return root;

}

else if (root->key < k) {

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

return root;

}

// We reach here when root is the node

// to be deleted.

// If one of the children is empty

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

}

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

Node\* temp = root->left;

delete root;

return temp;

}

// If both children exist

else {

Node\* succParent = root;

// Find successor

Node\* succ = root->right;

while (succ->left != NULL) {

succParent = succ;

succ = succ->left;

}

// Delete successor. Since successor

// is always left child of its parent

// we can safely make successor's right

// right child as left of its parent.

// If there is no succ, then assign

// succ->right to succParent->right

if (succParent != root)

succParent->left = succ->right;

else

succParent->right = succ->right;

// Copy Successor Data to root

root->key = succ->key;

// Delete Successor and return root

delete succ;

return root;

}

}

// Driver Code

int main()

{

/\* Let us create following BST

50

/ \

30 70

/ \ / \

20 40 60 80 \*/

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);

printf("Original BST: ");

inorder(root);

printf("\n\nDelete a Leaf Node: 20\n");

root = deleteNode(root, 20);

printf("Modified BST tree after deleting Leaf Node:\n");

inorder(root);

printf("\n\nDelete Node with single child: 70\n");

root = deleteNode(root, 70);

printf("Modified BST tree after deleting single child Node:\n");

inorder(root);

printf("\n\nDelete Node with both child: 50\n");

root = deleteNode(root, 50);

printf("Modified BST tree after deleting both child Node:\n");

inorder(root);

return 0;

}