Binary Search Tree Deletion

When deleting a node in a BST, we must **maintain the BST property.**
There are **3 cases** to consider:

Case 1 – Node has No Child (Leaf Node)
Case 2 – Node has One Child
Case 3 – Node has Two Children

Code Structure

```
#include <iostream>
using namespace std;

class Node {
public:
    int key;
    Node* left;
    Node* right;

    Node(int value) {
        key = value;
        left = right = nullptr;
    }
};

// Insert a node
Node* insert(Node* root, int key) {
    if (root == nullptr) {
        return new Node(key);
    }
    if (key < root->key) {
        root->left = insert(root->left, key);
    } else {
        root->right = insert(root->right, key);
    }
    return root;
}
```

```
// Inorder Traversal
void inorder(Node* root) {
    if (root == nullptr) {
        return;
    }
    inorder(root->left);
    cout << root->key << " ";
    inorder(root->right);
}

// Function to delete a node in BST
Node* deleteNode(Node* root, int key) {
    if (root == nullptr)
        return root;

    if (key < root->key) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->key) {
        root->right = deleteNode(root->right, key);
    } else {
        // Node to be deleted found

        // Case 1: No child
        if (root->left == nullptr && root->right == nullptr) {
            delete root;
            return nullptr;
        }

        // Case 2: One child
        else if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
```

```
        // Case 3: Two children
        else {
            // Find inorder successor (smallest in right subtree)
            Node* temp = root->right;
            while (temp && temp->left != nullptr) {
                temp = temp->left;
            }

            // Replace root with inorder successor
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    return root;
}

// Main function
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    cout << "Binary Search Tree (Inorder): ";
    inorder(root);
    cout << endl;

    int keyToDelete = 50;
    root = deleteNode(root, keyToDelete);

    cout << "BST after deleting " << keyToDelete << ": ";
    inorder(root);
    cout << endl;

    return 0;
}
```

Output-

Binary Search Tree (Inorder): 20 30 40 50 60 70 80
BST after deleting 50: 20 30 40 60 70 80