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

struct Node {
    int data;
    Node* left;
    Node* right;
};

// Cipta node baru
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

// Semak jika BST kosong
bool IsEmpty(Node* root) {
    return root == nullptr;
}

// Masukkan nilai dalam BST
Node* Insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }

    if (value < root->data) {
        root->left = Insert(root->left, value);
    } else {
        root->right = Insert(root->right, value);
    }

    return root;
}

// Inorder traversal: kiri - root - kanan
void Print(Node* root) {
    if (root != nullptr) {
        Print(root->left);
        cout << root->data << " ";
        Print(root->right);
    }
}

// Cari jumlah node dalam BST
int CountNodes(Node* root) {
    if (root == nullptr) return 0;
    return 1 + CountNodes(root->left) + CountNodes(root->right);
}

// Cari nilai dalam BST
```

```
bool Search(Node* root, int item) {
    if (root == nullptr) return false;
    if (item == root->data) return true;
    else if (item < root->data) return Search(root->left, item);
    else return Search(root->right, item);
}

// Cari nilai minimum dalam BST
int FindMin(Node* root) {
    if (root == nullptr) {
        cout << "Tree is empty.\n";
        return -1;
    }
    while (root->left != nullptr) {
        root = root->left;
    }
    return root->data;
}

// Cari nilai maksimum dalam BST
int FindMax(Node* root) {
    if (root == nullptr) {
        cout << "Tree is empty.\n";
        return -1;
    }
    while (root->right != nullptr) {
        root = root->right;
    }
    return root->data;
}

// Padam node dari BST
Node* Delete(Node* root, int item) {
    if (root == nullptr) return root;

    if (item < root->data) {
        root->left = Delete(root->left, item);
    } else if (item > root->data) {
        root->right = Delete(root->right, item);
    } else {
        // 0 atau 1 anak
        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;
        }

        // 2 anak - guna nilai terkecil dari subtree kanan
        int minValue = FindMin(root->right);
        root->data = minValue;
```

```
        root->right = Delete(root->right, minValue);
    }

    return root;
}

// ================================
// Fungsi utama untuk test BST
// ================================

int main() {
    Node* root = nullptr;

    // Insert
    root = Insert(root, 8);
    Insert(root, 3);
    Insert(root, 10);
    Insert(root, 1);
    Insert(root, 6);
    Insert(root, 14);
    Insert(root, 4);
    Insert(root, 7);

    cout << "Inorder Print (BST): ";
    Print(root);
    cout << endl;

    cout << "Total nodes: " << CountNodes(root) << endl;

    cout << "Find 6? " << (Search(root, 6) ? "Yes" : "No") << endl;
    cout << "Find 20? " << (Search(root, 20) ? "Yes" : "No") << endl;

    cout << "Minimum: " << FindMin(root) << endl;
    cout << "Maximum: " << FindMax(root) << endl;

    cout << "Delete 3" << endl;
    root = Delete(root, 3);
    cout << "Inorder after delete: ";
    Print(root);
    cout << endl;

    cout << "Tree is " << (IsEmpty(root) ? "empty" : "not empty") << endl;

    return 0;
}
```