BINARY SEARCH TREE

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

// Node class
class Node {
public:
    int data;
    Node* left;
    Node* right;

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

// Insert function
Node* insert(Node* root, int value) {
    if (root == nullptr)
        return new Node(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Inorder traversal
void inorder(Node* root) {
    if (root == nullptr) return;

    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = nullptr;

    // Insert elements
    int arr[] = {32, 35, 36, 38, 30, 37};
    int n = 6;

    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);

    cout << "Inorder Traversal: ";
    inorder(root);

    return 0;
      }
```

BST with ALL Traversals (Inorder, Preorder, Postorder)

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

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

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

Node* insert(Node* root, int value) {
    if (root == nullptr)
        return new Node(value);
```

```cpp
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

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

// Preorder (NLR)
void preorder(Node* root) {
    if (root == nullptr) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

// Postorder (LRN)
void postorder(Node* root) {
    if (root == nullptr) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main() {
    Node* root = nullptr;

    int arr[] = {32, 35, 36, 38, 30, 37};
    int n = 6;

    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);

    cout << "\nInorder   : ";
    inorder(root);
    cout << "\nPreorder  : ";
    preorder(root);
    cout << "\nPostorder : ";
    postorder(root);

    return 0;
}
```

BST with Searching an Element

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

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

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

Node* insert(Node* root, int value) {
    if (root == nullptr)
        return new Node(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Search function
bool search(Node* root, int key) {
    if (root == nullptr)
        return false;

    if (root->data == key)
        return true;

    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}
int main() {
    Node* root = nullptr;
    int arr[] = {32, 35, 36, 38, 30, 37};
    int n = 6;

    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);

    int key;
    cout << "Enter element to search: ";
    cin >> key;

    if (search(root, key))
        cout << key << " found in BST\n";
    else
        cout << key << " NOT found in BST\n";
    return 0;
  }
```

<u>Code for BST + Deletion</u>

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

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

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

Node* insert(Node* root, int value) {
    if (root == nullptr)
        return new Node(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Find minimum node (for deletion)
Node* findMin(Node* root) {
    while (root->left != nullptr)
        root = root->left;
    return root;
}

// Delete function
Node* deleteNode(Node* root, int key) {
    if (root == nullptr)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // Case 1: No child
        if (root->left == nullptr && root->right == nullptr) {
            delete root;
            return nullptr;
        }

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

        // Case 3: Two children
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inorder(Node* root) {
    if (root == nullptr) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = nullptr;

    int arr[] = {32, 35, 36, 38, 30, 37};
    int n = 6;

    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);

    cout << "Inorder before deletion: ";
    inorder(root);

    int key;
    cout << "\nEnter element to delete: ";
    cin >> key;

    root = deleteNode(root, key);

    cout << "Inorder after deletion: ";
    inorder(root);

    return 0;
}
```

Check if a Binary Tree is a BST

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

class Node {
public:
    int data;
    Node* left;
```

```cpp
    Node* right;

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

// Function to check if tree is BST
bool isBST(Node* root, int minVal, int maxVal) {
    if (root == nullptr)
        return true;

    if (root->data <= minVal || root->data >= maxVal)
        return false;

    return isBST(root->left, minVal, root->data) &&
           isBST(root->right, root->data, maxVal);
}

int main() {
    // Manually create a tree

    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(20);
    root->left->left = new Node(2);
    root->left->right = new Node(8);

    if (isBST(root, INT_MIN, INT_MAX))
        cout << "It is a BST";
    else
        cout << "Not a BST";

    return 0;
}
```