Binary Search Tree – Project Report

Name: Umer Bin Hussain  
Class: Intermediate  
Project Title: Binary Search Tree Implementation in C++  
Language: C++  
IDE: Code::Blocks / Dev C++ / Visual Studio  
Compiler: g++ (MinGW)

# Table of Contents

1. Introduction

2. Objectives

3. Features Implemented

4. Tools & Technologies

5. Complete Code

6. Output Screenshots

7. Conclusion

# 1. Introduction

This project implements a Binary Search Tree (BST) using C++. It supports standard operations like insert, delete, search, and four types of tree traversals. The purpose is to understand recursive functions, tree structures, and object-oriented programming concepts.

# 2. Objectives

• Understand Binary Search Tree (BST) structure.

• Implement core BST operations: Insert, Delete, Search.

• Display data using Inorder, Preorder, Postorder, and Level-order Traversals.

• Practice Object-Oriented Programming concepts (Encapsulation, Abstraction, etc.)

# 3. Features Implemented

Insert Data: Insert 5 values into the BST.

Delete Data: Deletes a specific node if it exists.

Search: Search if a number exists in the BST.

Traversals: Inorder, Preorder, Postorder, and Level-order traversals.

Menu-Driven Console: Interactive menu to perform operations.

OOP Concepts Used: Classes, Objects, Encapsulation, Inheritance-ready structure.

# 4. Tools & Technologies

• Programming Language: C++

• Compiler: GCC (g++)

• IDE Used: Code::Blocks / Dev C++

• Libraries: iostream, cstdlib, algorithm, queue, limits

# 5. Complete Source Code

The following is the complete C++ source code of the BST project:

#include<iostream>

#include <cstdlib>

#include<limits> #include <queue> **using namespace std**;

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

Node\* right; Node(**int** value) : data(value), left(NULL), right(NULL) {} };

**class** BST { **public**: Node\* root;

BST() { root = NULL; } **private**:

**bool search**(Node\* root,**int** data){ **if**(root==NULL){ **return false**; }

**else if**(root->data==data) **return true**; **else if**(root->data>data) **return search**(root->left,data); **else**

**return search**(root->right,data); } Node\* **insert**(Node\* root, **int** data) { **if** (root == NULL) { **return new** Node(data); }

**if** (data < root->data) { root->left = **insert**(root->left, data); } **else** {

root->right = **insert**(root->right, data); } **return** root; }

Node\* findMin(Node\* node) { **while** (node && node->left != NULL) { node = node->left; } **return** node; } Node\* deleteNode(Node\* root, **int** data) { **if** (root == NULL) **return** root;

**if** (data < root->data) {

root->left = deleteNode(root->left, data); }

**else if** (data > root->data) { root->right = deleteNode(root->right, data); } **else** {

**if** (root->left == NULL && root->right == NULL) { **delete** root; **return** NULL; }

**else 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; } **else** { Node\* temp = findMin(root->right); root->data = temp->data;

root->right = deleteNode(root->right, temp->data); } } **return** root; }

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

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

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

**void** levelOrder(Node\* root) { **if** (root == NULL) **return**; **queue**<Node\*> q;

q.push(root); **while** (!q.**empty**()) { Node\* current = q.**front**();

q.pop();

**cout** << current->data << " "; **if** (current->left) q.push(current->left); **if** (current->right) q.push(current->right); }

} **public**:

**bool search**(**int** data){ **return search**(root,data); }

**void insert**(**int** data) { root = **insert**(root, data); }

**void** printpostorder(){ postorder(root); **cout**<<**endl**; } **void** printInorder() { inorder(root); **cout** << **endl**; }

**void** printPreorder() { preorder(root); **cout** << **endl**;

}

**void** printlevelorder(){ levelOrder(root); **cout**<<**endl**; }

**void** deleteData(**int** data) { root = deleteNode(root, data); } };

**void** waitForEnter() {

**cout** << "\nPress Enter to continue..."; **cin**.ignore(**numeric\_limits**<streamsize>::**max**(), '\n'); **cin**.get(); } **int** main() {

BST tree; **int** num; **int** choice; **bool** run=**false**; **do**{

**cout**<<"==== BINARY SEACRH TREE ====\n";

**cout**<<"1:Insert data\n2:Delete data\n3:Search data\n4:In-order traversal\n5:Post-order traversal\n6:pre-order traversal\n7:level-order traversal\n8:Exit\n"; **cout**<<"Enter choice:"; **cin**>>choice; **switch**(choice){ **case** 1:{ **int** MAX;

**cout**<<"How many numbers you want to enter:"; **cin**>>MAX; **if**(MAX<=0){

**cout**<<"enter number greater than 0"; **break**; }

**int** number[MAX];

**cout**<<"Enter "<<MAX<<" number:"; **for**(**int** i=0;i<MAX;i++){ **cin**>>number[i]; ; }

**for** (**int** i = 0; i < MAX; i++) { tree.**insert**(number[i]); } run = **true**; **break**;} **case** 2:{ **if**(run==**true**){

**cout**<<"Enter data you want to delete: "; **cin**>>num; **if**(tree.**search**(num)){ tree.deleteData(num); **cout**<<"data deleted\n";} **else**

**cout**<<"data not found\n";

**break**; } **else**

**cout**<<"Insert Numbers first\n"; **break**;} **case** 3:{ **if**(run==**true**){ **cout**<<"search number: ";

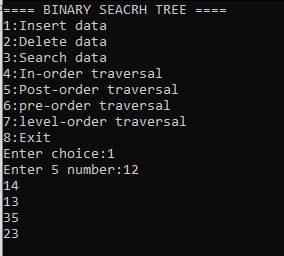
**cin**>>num; **if**(tree.**search**(num)) **cout**<<num<<" is found\n"; **else cout**<<num<<" not found\n";} **else cout**<<"Insert Numbers first\n";

**break**;} **case** 4:{ **if**(run==**true**){ **cout** << "Inorder Traversal: "; tree.printInorder();} **else cout**<<"Insert Numbers first\n"; **break**;} **case** 5:{ **if**(run==**true**){ **cout** << "Post-order Traversal: "; tree.printpostorder();} **else cout**<<"Insert Numbers first\n"; **break**;} **case** 6:{ **if**(run==**true**){ **cout** << "Pre-order Traversal: "; tree.printPreorder();} **else cout**<<"Insert Numbers first\n"; **break**;} **case** 7:{ **if**(run==**true**){ **cout** << "Level-order Traversal: "; tree.printlevelorder();} **else cout**<<"Insert Numbers first\n"; **break**;} **case** 8: **return** 0; **default**: **cout**<<"Enter correct choice\n"; **break**; } waitForEnter(); system("cls"); }**while**(1);

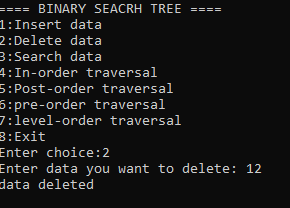
}

# 6. Output Screenshots

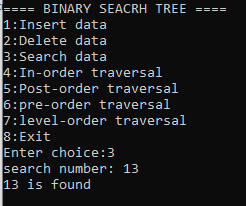
▶ Insertion



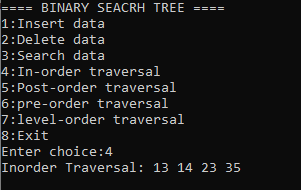
▶ Deletion



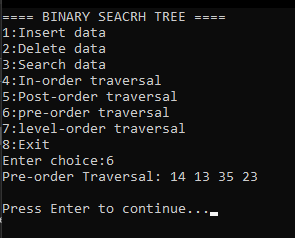
▶ Search



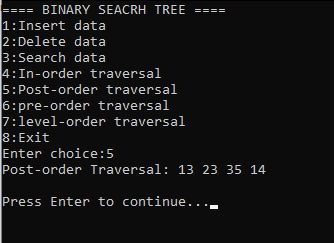
▶ Inorder Traversal



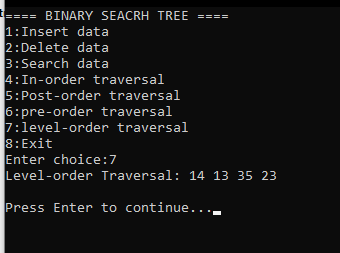
▶ Preorder Traversal



▶ Postorder Traversal



▶ Level-order Traversal



# 7. Conclusion

This project helped in deeply understanding how a Binary Search Tree works. Through recursive implementation and traversal logic, I learned how data structures can efficiently manage and organize data. The use of C++ and object-oriented programming allowed a clean, scalable implementation.