Computer Networks Laboratory
(CSDC-0236)

B.Tech IVth Semester
(January – June 2025)

Submitted by

Ritik Gupta(23103122)
Group-G2

Submitted to

Dr. Samayveer Singh

Department of Computer Science & Engineering
Dr. B. R. Ambedkar National Institute of Technology Jalandhar -
144008, Punjab, India

ASSIGNMENT – 05

```
// C++ Program for Implementing Binary Tree
#include <iostream>
#include <queue>
using namespace std;

// Template class for the Node of a Binary Tree
template <typename T>
class Node {
public:
    // Data held by the node
    T data;
    // Pointer to the left child
    Node* left;
    // Pointer to the right child
    Node* right;

    // Constructor to initialize the node with a value
    Node(T value) : data(value), left(nullptr), right(nullptr) {}
};

// Template class for a Binary Tree
template <typename T>
class BinaryTree {
private:
    // Pointer to the root of the tree
    Node<T>* root;

    // Recursive Function to delete a node from the tree
    Node<T>* deleteRecursive(Node<T>* current, T value) {
        if (current == nullptr) return nullptr;

        if (current->data == value) {
            if (current->left == nullptr && current->right == nullptr) {
                delete current;
                return nullptr;
            }
            if (current->left == nullptr) {
                Node<T>* temp = current->right;
                delete current;
                return temp;
            }
            if (current->right == nullptr) {
                Node<T>* temp = current->left;
                delete current;
                return temp;
            }

            Node<T>* successor = findMin(current->right);
            current->data = successor->data;
            current->right = deleteRecursive(current->right, successor->data);
        } else {
            current->left = deleteRecursive(current->left, value);
            current->right = deleteRecursive(current->right, value);
        }
        return current;
    }

    // Helper Function to find the minimum value node
    Node<T>* findMin(Node<T>* node) {
        while (node->left != nullptr) node = node->left;
        return node;
    }

    // Recursive Function to search for a value in the tree
    bool searchRecursive(Node<T>* current, T value) {
        if (current == nullptr) return false;
        if (current->data == value) return true;
        return searchRecursive(current->left, value) ||
          searchRecursive(current->right, value);
    }

    // Function for Recursive inorder traversal of the tree
    void inorderRecursive(Node<T>* node) {
        if (node != nullptr) {
            inorderRecursive(node->left);
            cout << node->data << " ";
            inorderRecursive(node->right);
        }
    }

    // Function for Recursive preorder traversal of the tree
    void preorderRecursive(Node<T>* node) {
        if (node != nullptr) {
            cout << node->data << " ";
            preorderRecursive(node->left);
            preorderRecursive(node->right);
        }
    }

    // Function for Recursive postorder traversal of the tree
    void postorderRecursive(Node<T>* node) {
        if (node != nullptr) {
            postorderRecursive(node->left);
            postorderRecursive(node->right);
            cout << node->data << " ";
        }
    }

public:
    // Constructor to initialize the tree
    BinaryTree() : root(nullptr) {}

    // Function to insert a node in the binary tree
    void insertNode(T value) {
        Node<T>* newNode = new Node<T>(value);

        if (root == nullptr) {
            root = newNode;
            return;
        }
        queue<Node<T>*> q;
        q.push(root);

        while (!q.empty()) {
            Node<T>* current = q.front();
            q.pop();

            if (current->left == nullptr) {
                current->left = newNode;
                return;
            } else {
                q.push(current->left);
            }

            if (current->right == nullptr) {
                current->right = newNode;
                return;
            } else {
                q.push(current->right);
            }
        }
    }

    // Function to delete a node from the tree
    void deleteNode(T value) {
        root = deleteRecursive(root, value);
    }

    // Function to search for a value in the tree
    bool search(T value) {
        return searchRecursive(root, value);
    }

    // Function to perform inorder traversal of the tree
    void inorder() {
        inorderRecursive(root);
        cout << endl;
    }

    // Function to perform preorder traversal of the tree
    void preorder() {
        preorderRecursive(root);
        cout << endl;
    }

    // Function to perform postorder traversal of the tree
    void postorder() {
        postorderRecursive(root);
        cout << endl;
    }

    // Function  to perform level order traversal of the tree
    void levelOrder() {
        if (root == nullptr) return;

        queue<Node<T>*> q;
        q.push(root);

        while (!q.empty()) {
            Node<T>* current = q.front();
            q.pop();

            cout << current->data << " ";

            if (current->left != nullptr) q.push(current->left);
            if (current->right != nullptr) q.push(current->right);
        }
        cout << endl;
    }
};

int main() {
    BinaryTree<int> tree;
```

p1.cpp (continued, Page 6/6)

```
    // Insert the nodes into the tree
    tree.insertNode(1);
    tree.insertNode(2);
    tree.insertNode(3);
    tree.insertNode(4);
    tree.insertNode(5);
    tree.insertNode(6);

    cout << "Inorder traversal: ";
    tree.inorder();

    cout << "Preorder traversal: ";
    tree.preorder();

    cout << "Postorder traversal: ";
    tree.postorder();

    cout << "Level order traversal: ";
    tree.levelOrder();

    cout << "Searching for 7: " << (tree.search(7) ? "Found" : "Not
      Found") << endl;
    cout << "Searching for 6: " << (tree.search(6) ? "Found" : "Not
      Found") << endl;

    tree.deleteNode(3);
    cout << "Inorder traversal after removing 3: ";
    tree.inorder();

    return 0;
}
```

Output of p1.cpp (Page 1/1)

```
Inorder traversal: 4 2 5 1 6 3
Preorder traversal: 1 2 4 5 3 6
Postorder traversal: 4 5 2 6 3 1
Level order traversal: 1 2 3 4 5 6
Searching for 7: Not Found
Searching for 6: Found
Inorder traversal after removing 3: 4 2 5 1 6
```

File: p1.py (Page 1/1)

```
print("Why not")
```

Output of p1.py (Page 1/1)

```
Why not
```

File: p2.cpp (Page 1/1)

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    for(int i=0;i<50;i++){
        cout<<"Praveen\n";
    }
}
```

Output of p2.cpp (Page 1/2)

```
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
```

Output of p2.cpp (continued, Page 2/2)

```
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
Praveen
```