**Experiment 7**

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C013 | Name: Ashmit Jain |
| Class: B | Batch: B1 |
| Date of Experiment: | Date of Submission: 22/9/2024 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

#include <iostream>

using namespace std;

class Node {

public:

    int data;         // Data stored in the node

    Node\* left;       // Pointer to the left child

    Node\* right;      // Pointer to the right child

    // Constructor to initialize a node with a given value

    Node(int value) {

        data = value;

        left = nullptr;

        right = nullptr;

    }

    // Pre-order traversal (root, left, right)

    void preOrderTraversal(Node\* node) {

        if (node == nullptr) return;

        cout << node->data << " ";

        preOrderTraversal(node->left);

        preOrderTraversal(node->right);

    }

};

int main() {

    // Manually constructing the binary tree

    Node\* root = new Node(50);

    root->left = new Node(30);

    root->right = new Node(70);

    root->left->left = new Node(20);

    root->left->right = new Node(40);

    root->right->left = new Node(60);

    root->right->right = new Node(80);

    // Pre-order traversal of the tree

    cout << "Pre-order traversal: ";

    root->preOrderTraversal(root);

    cout << endl;

    // Cleaning up memory

    delete root;

    return 0;

}

**Task2:**

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node\* left;

    Node\* right;

    // Constructor to initialize the node with data

    Node(int value = 0)

    {

        data = value;

        left = nullptr;

        right = nullptr;

    }

    // Destructor to clean up dynamically allocated memory

    ~Node()

    {

        delete left;

        delete right;

    }

    // Pre-order traversal (root, left, right)

    void preOrderTraversal(Node\* root)

    {

        if (root == nullptr) return;

        cout << root->data << " ";

        preOrderTraversal(root->left);

        preOrderTraversal(root->right);

    }

    // In-order traversal (left, root, right)

    void inOrderTraversal(Node\* root)

    {

        if (root == nullptr) return;

        inOrderTraversal(root->left);

        cout << root->data << " ";

        inOrderTraversal(root->right);

    }

    // Post-order traversal (left, right, root)

    void postOrderTraversal(Node\* root)

    {

        if (root == nullptr) return;

        postOrderTraversal(root->left);

        postOrderTraversal(root->right);

        cout << root->data << " ";

    }

};

int main()

{

    // Initialize the main root of the tree

    Node\* root = new Node(10);

    // Manually build the tree

    root->left = new Node(2);

    root->right = new Node(3);

    root->left->left = new Node(4);

    root->left->right = new Node(5);

    root->right->left = new Node(6);

    root->right->right = new Node(7);

    // Traversals

    cout << "Pre-order traversal: ";

    root->preOrderTraversal(root);

    cout << endl;

    cout << "In-order traversal: ";

    root->inOrderTraversal(root);

    cout << endl;

    cout << "Post-order traversal: ";

    root->postOrderTraversal(root);

    cout << endl;

    // Cleanup memory

    delete root;

    return 0;

}

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

****

**Task2:**

****

**B.3 Observations and learning [w.r.t. all tasks]:**

**Representation of Binary Tree**:

Binary trees can be represented in memory using either arrays or linked lists.

The linked list representation is more efficient for binary trees because it dynamically allocates memory for each node, allowing non-contiguous storage of nodes in memory.

**Binary Tree Structure**:

Each node in a binary tree contains three components: data, a left pointer, and a right pointer.

The left pointer references the left child, the right pointer references the right child, and if a node has no children, the pointers are set to nullptr.

**Tree Traversal**:

**Preorder Traversal**:

The root is visited first, followed by the left subtree, then the right subtree.

**Inorder Traversal**:

The left subtree is visited first, followed by the root, then the right subtree.

**Postorder Traversal**:

The left subtree is visited first, followed by the right subtree, and the root is visited last.

**DFS Traversals**:

The Depth-First Search (DFS) method allows efficient tree traversal by exploring nodes down a branch before backtracking. All three traversals are forms of DFS, differing only in the order nodes are visited.

**B.4 Conclusion:**

**Memory Representation**:

The linked list representation of binary trees allows flexible memory allocation and easy manipulation of nodes, making it a preferred method over array representation for binary trees.

**Traversal Algorithms**:

Implementing DFS traversal techniques (preorder, inorder, and postorder) provides a comprehensive method to access and process all nodes in a binary tree. These traversal techniques are foundational in tree operations and are useful in various applications like expression trees, syntax trees, and searching.

**Binary Tree Analysis**:

Understanding tree height, node relationships, and traversal methods is critical for efficient manipulation and navigation of binary tree structures. Traversals provide different views of tree data, and each method is suitable for specific use cases in programming.