ACTIVITY NO. 9.2

Implementing and Traversing Binary Trees	
Course Code: CPE009B	**Program:** Computer Engineering
Course Title: Object Oriented Programming	**Date Performed:** November 27, 2024
Section: CPE21S4	**Date Submitted:** November 27, 2024
Name: **-** ESTEBAN, Prince Wally G. - FERNANDEZ, Don Eleazar T. - SANCHEZ, Christan Ray R. - TANDAYU, Leoj Jeam B. - VALLESER, Kenn L.	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s)

- To implement a binary search tree to solve a given problem
- To perform different traversal methods and analyze the different outputs

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

- Implement the tree data structure to solve given problems.

3. Discussion

Refer to the following topics in order to understand the presented activity:

4.1 Trees

4.1.1 Parse Tree; Tree Traversals

4.1.2 Priority Queues with Binary Heaps (w/ Example, Operations, and Implementation)

4.1.3 Binary Search Trees (Operations, Implementation, Analysis)

4.1.4 Balanced BST

In-order traversal visits the left subtree first, then the root, and finally the right subtree, making it particularly useful for binary search trees (BSTs) as it processes the nodes in ascending order. Pre-order traversal, which processes the root before its subtrees, is often used in tasks like copying trees or evaluating expressions. Post-order traversal, which visits the left and right subtrees before the root, is helpful in operations like tree deletion or evaluating postfix expressions. The article also explains both recursive and iterative implementations of these traversals, with recursion being the most intuitive approach due to the hierarchical nature of trees, while iterative methods use stacks to simulate recursion. This comprehensive explanation helps readers understand how to choose the right traversal method for specific applications in data structures and algorithms.

4. Materials and Equipment

- Personal Computer w/ C++ Compiler

5. Procedure

1. Write a program that will implement the use of a Tree. The program will allow the user to perform the inorder, preorder and the postorder traversal.

2. Based on your output. Create a diagram to show the tree after all values have been inserted. Then, with the use of visual aids (like arrows and numbers) indicate the traversal order for in-order, pre-order and post-order traversal on the diagram.

(Screenshot of tree diagram)

(Screenshot of tree diagram with indicated in-order traversal)

(Screenshot of tree diagram with indicated pre-order traversal)

(Screenshot of tree diagram with indicated post-order traversal)

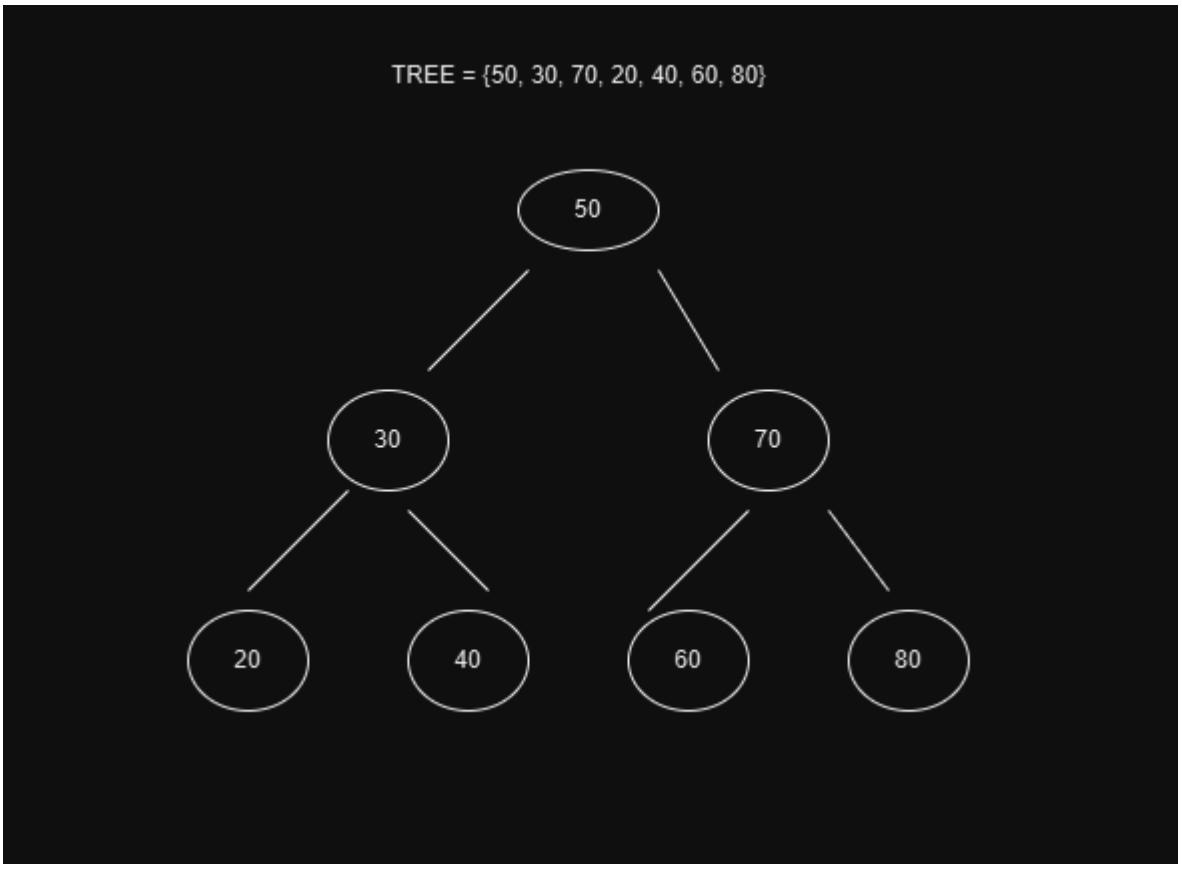
6. Output

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

class Node {
public:
    int value;
    Node* left;
    Node* right;
    Node(int val) {
        value = val;
        left = right = nullptr;
    }
};

class BinaryTree {
public:
    Node* root;
    BinaryTree() {
        root = nullptr;
    }

    void insert(int value) {
        if (root == nullptr) {
            root = new Node(value);
```

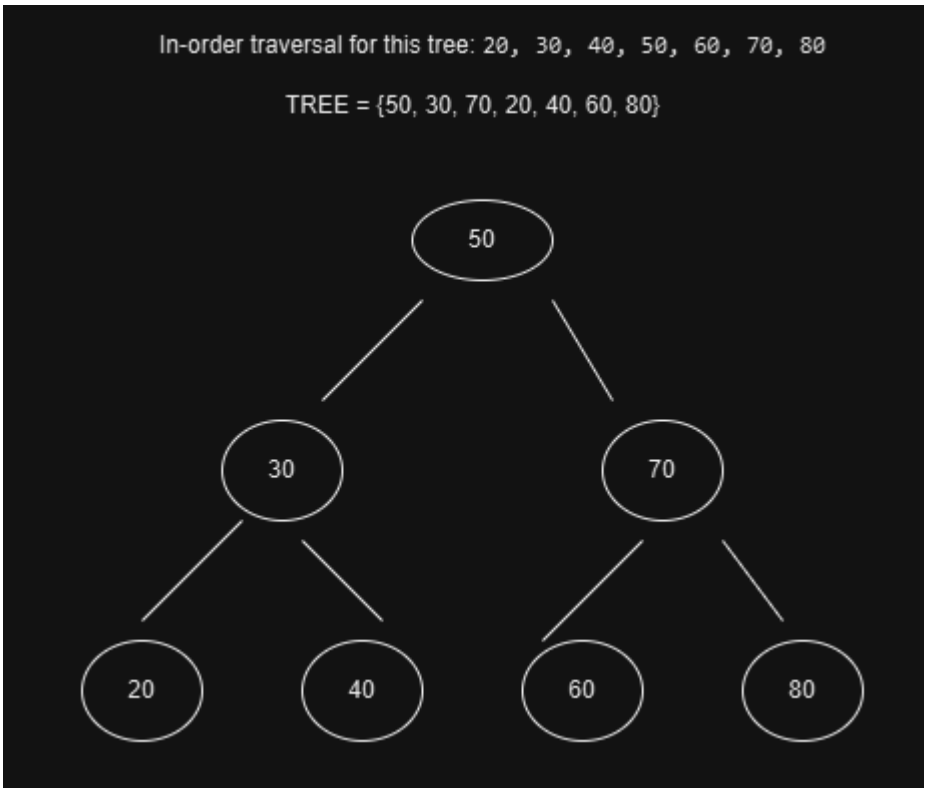
```
    } else {
        insertRecursive(root, value);
    }
}

void insertRecursive(Node* node, int value) {
    if (value < node->value) {
        if (node->left == nullptr) {
            node->left = new Node(value);
        } else {
            insertRecursive(node->left, value);
        }
    } else if (value > node->value) {
        if (node->right == nullptr) {
            node->right = new Node(value);
        } else {
            insertRecursive(node->right, value);
        }
    }
}

void inorder(Node* node) {
    if (node != nullptr) {
        inorder(node->left);
        cout << node->value << " ";
```

```
        inorder(node->right);
    }
}

void preorder(Node* node) {
    if (node != nullptr) {
        cout << node->value << " ";
        preorder(node->left);
        preorder(node->right);
    }
}

void postorder(Node* node) {
    if (node != nullptr) {
        postorder(node->left);
        postorder(node->right);
        cout << node->value << " ";
    }
}

void displayTraversals() {
    cout << "In-order Traversal: ";
    inorder(root);
    cout << "\nPre-order Traversal: ";
    preorder(root);
```

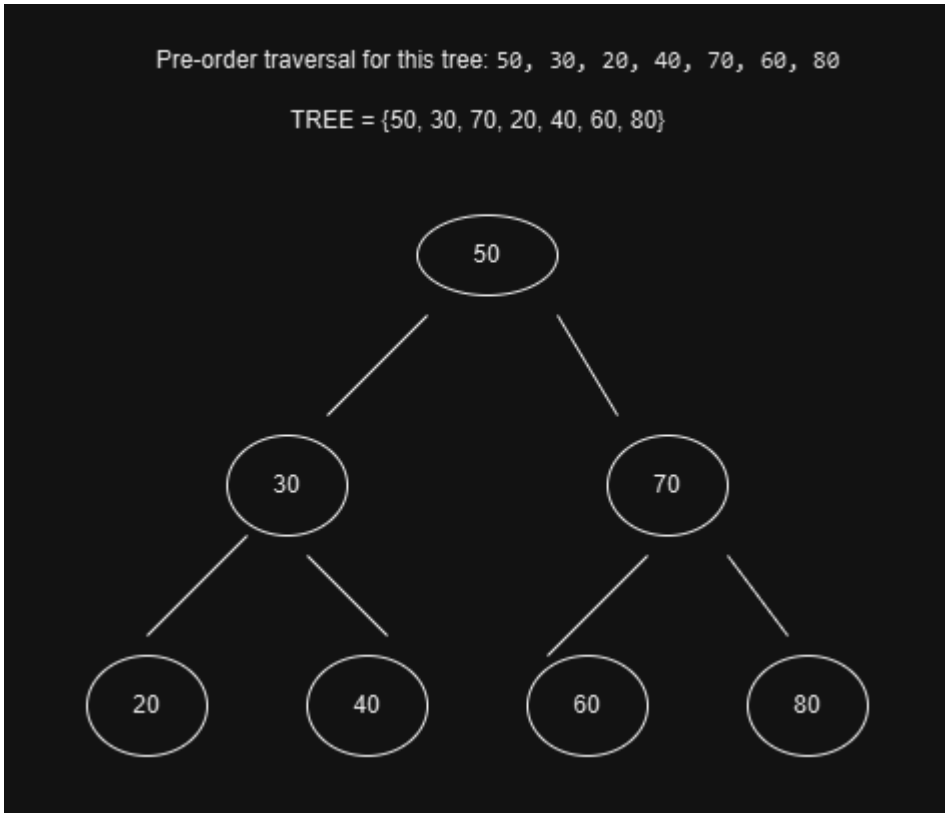
```
        cout << "\nPost-order Traversal: ";
        postorder(root);
        cout << endl;
    }
};

int main() {
    BinaryTree tree;
    int values[] = {50, 30, 70, 20, 40, 60, 80};
    for (int value : values) {
        tree.insert(value);
    }
    tree.displayTraversals();
    return 0;
}
```



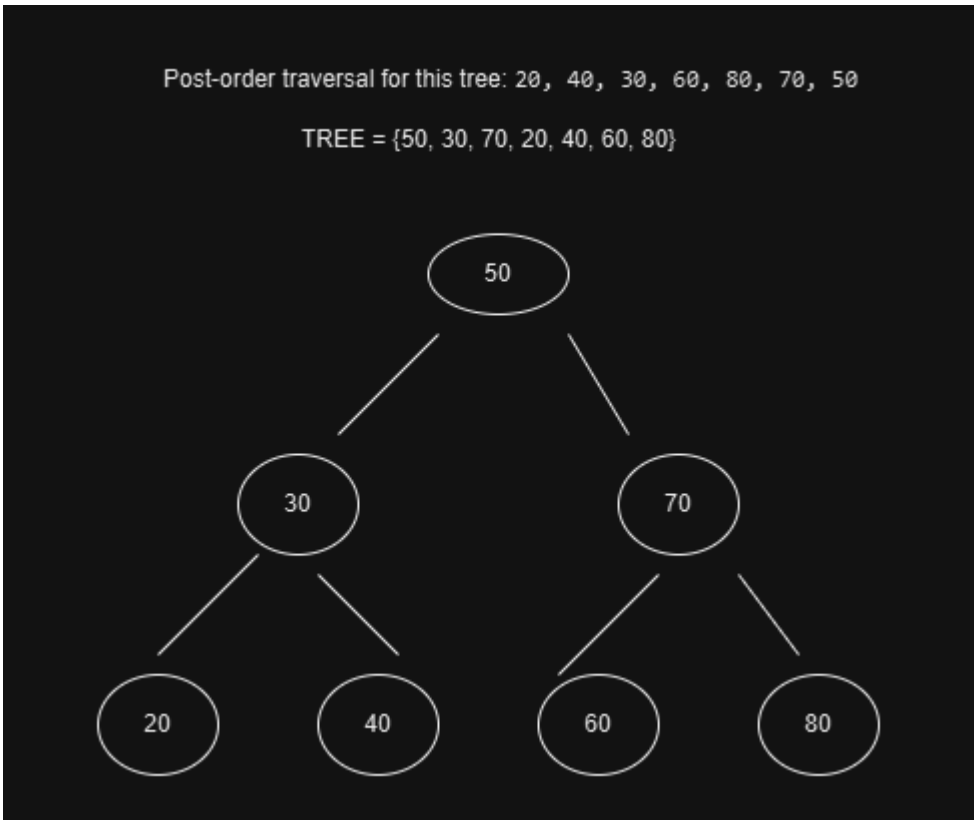
Tree Diagram 1.1



In-order traversal method 1.2



Pre-order traversal method 1.3



Post-order traversal for this tree 1.2

7. Conclusion

In this Activity, I implemented a Binary Search Tree (BST) in C++ and learned how to perform three types of tree traversals: In-order, Pre-order, and Post-order. Each traversal visits the nodes in a different order: In-order gives a sorted list, Pre-order starts with the root, and Post-order processes the children before the root. These traversal methods are useful in many real-life situations like sorting data, copying trees, or cleaning up memory. This exercise helped me understand how binary trees work and how different traversal techniques are applied in various scenarios.

8. References

- GeeksforGeeks, "Binary Tree Traversal," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/binary-tree-traversal/. [Accessed: 27-Nov-2024].

"I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own."