Activity No. 9.2	
Implementing and Traversing Binary Trees	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 27/11/2024
Section: CPE21S4	**Date Submitted:** 27/11/2024
Name(s): Mamaril, Justin Kenneth I, San Juan, Edson San Jose, Alexander Reyes, Alexzander Titong, Lee Ivan	**Instructor:** Prof. Maria Rizette Sayo

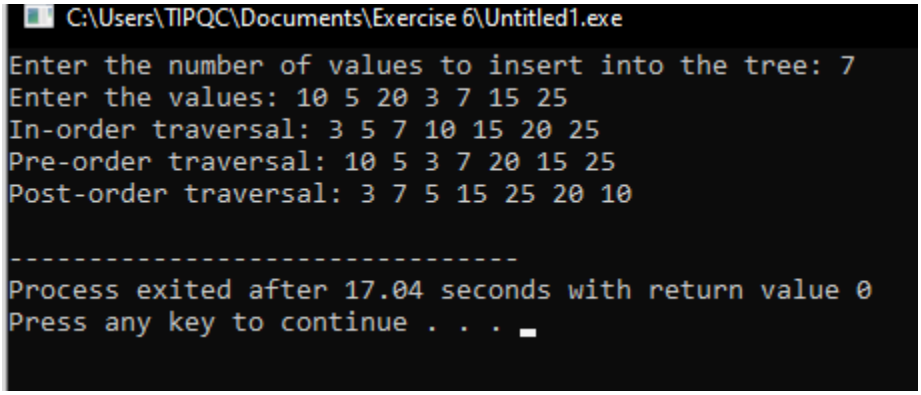
A. Procedure

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.

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

// Node structure for the binary tree
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Function to insert a node into the binary tree
Node* insertNode(Node* root, int value) {
    if (!root) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insertNode(root->left, value);
    } else {
        root->right = insertNode(root->right, value);
    }
    return root;
}

// In-order traversal (left, root, right)
void inOrderTraversal(Node* root) {
    if (root) {
        inOrderTraversal(root->left);
        cout << root->data << " ";
        inOrderTraversal(root->right);
    }
}

// Pre-order traversal (root, left, right)
void preOrderTraversal(Node* root) {
    if (root) {
        cout << root->data << " ";
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

// Post-order traversal (left, right, root)
void postOrderTraversal(Node* root) {
    if (root) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = nullptr;
    int n, value;

    cout << "Enter the number of values to insert into the tree: ";
    cin >> n;

    cout << "Enter the values: ";
    for (int i = 0; i < n; ++i) {
        cin >> value;
        root = insertNode(root, value);
```

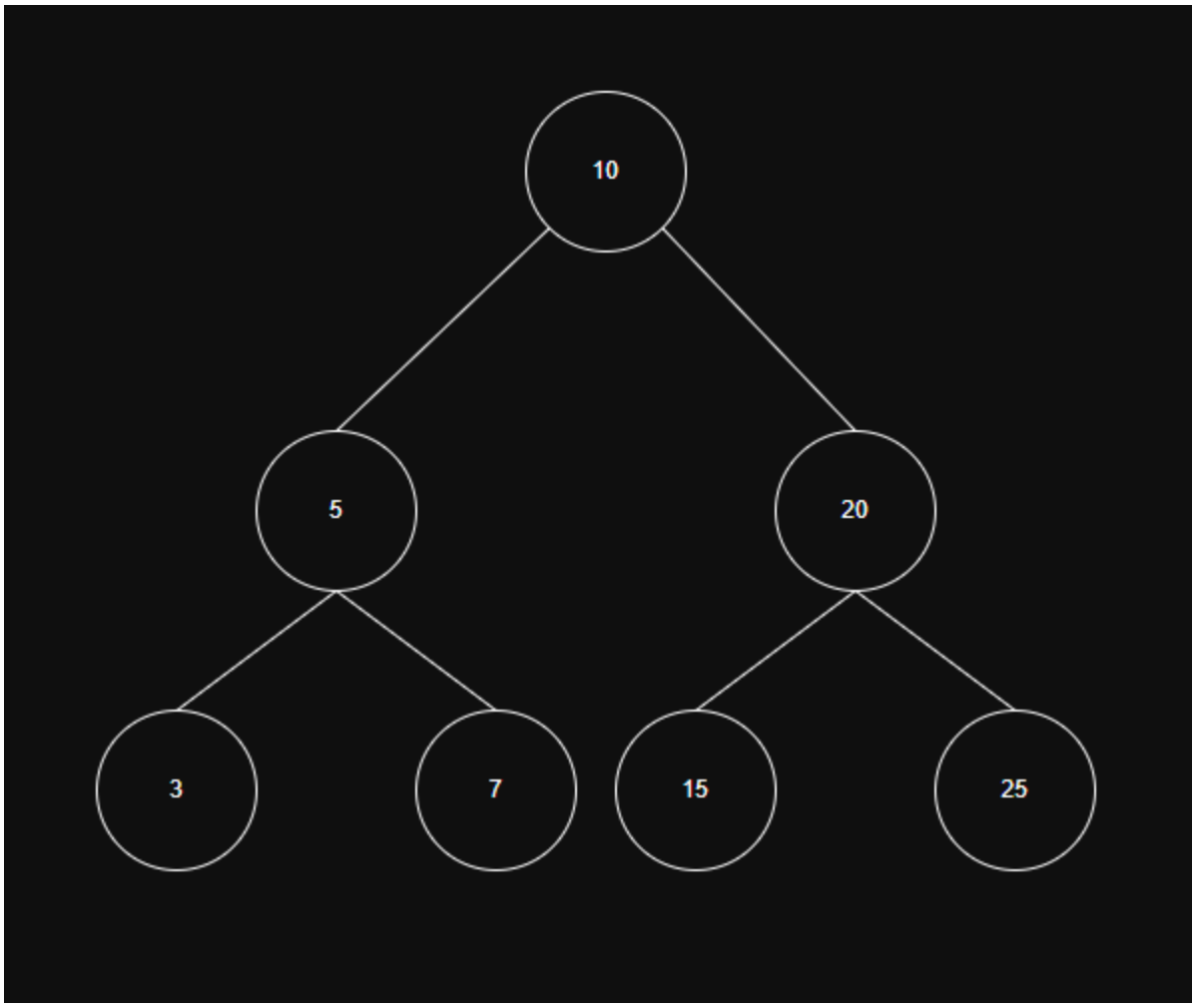
```
64      }
65
66      cout << "In-order traversal: ";
67      inOrderTraversal(root);
68      cout << endl;
69
70      cout << "Pre-order traversal: ";
71      preOrderTraversal(root);
72      cout << endl;
73
74      cout << "Post-order traversal: ";
75      postOrderTraversal(root);
76      cout << endl;
77
78      return 0;
79  }
80
```

OutPut:

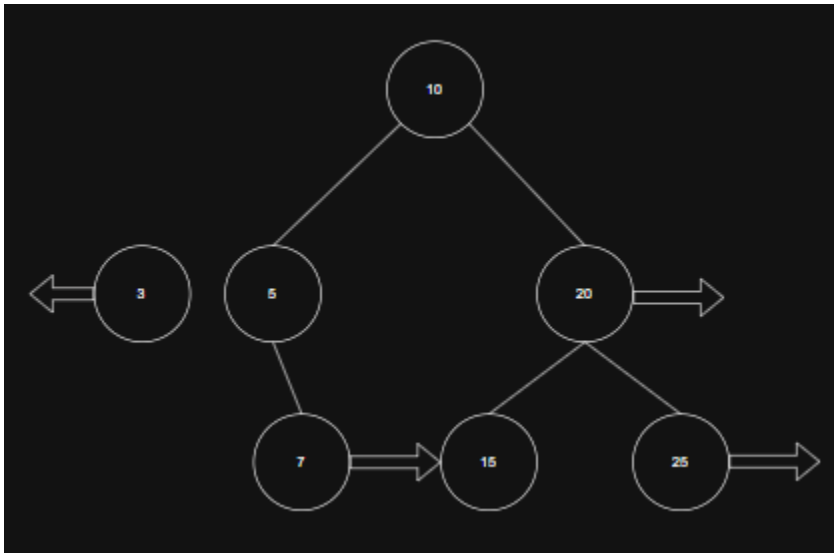


2. Based on your output. Create a diagram to show the tree after all values have been inserted. Then, with the use of visual aids (like arrows and numbers) indicate the traversal order for in-order, pre-order and post-order traversal on the diagram.

(Screenshot of tree diagram)

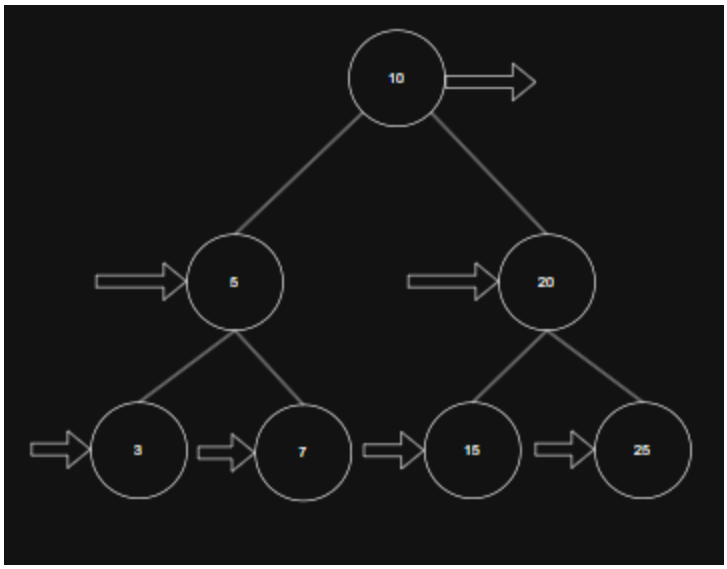


(Screenshot of tree diagram with indicated in-order traversal)



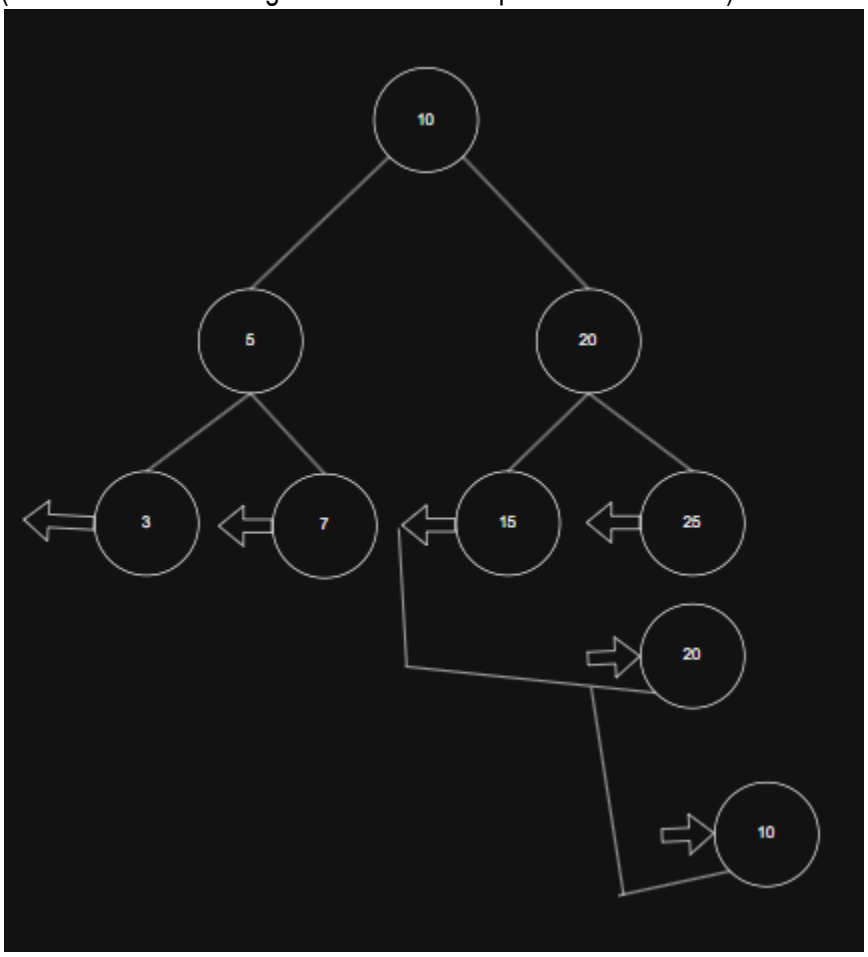
It starts at the leftmost node (3), moves up to its parent (5), and then visits the right child (7). Move to the root (10) and then traverse the right subtree: left child (15), root (20), and right child (25).

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



Visit the root (10), then move to the left subtree (5 → 3 → 7), and finally traverse the right subtree (20 → 15 → 25).

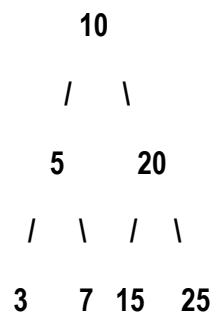
(Screenshot of tree diagram with indicated post-order traversal)



It starts with the left subtree (3 → 7 → 5), then moves to the right subtree (15 → 25 → 20), and finally visits the root (10).

B. Conclusion

The program demonstrates how to create a binary tree, insert values, and perform the three main types of traversals: in-order, pre-order, and post-order. Using the example input values, the program generates a tree structure like this:



From this tree, the outputs for each traversal are:

- **In-order: 3 → 5 → 7 → 10 → 15 → 20 → 25 (Left → Root → Right)**
- **Pre-order: 10 → 5 → 3 → 7 → 20 → 15 → 25 (Root → Left → Right)**
- **Post-order: 3 → 7 → 5 → 15 → 25 → 20 → 10 (Left → Right → Root)**

Using this output, diagrams of the tree were created, showing arrows and numbers to explain the traversal order for each type of traversal. These diagrams visually help to understand how the tree is processed during each traversal method.