Report on Magical Trees Assignment

This report presents the implementation and evaluation of the "Magical Trees" assignment, which involves understanding and implementing non-linear data structures using full binary trees. The objective is to determine whether two given trees satisfy the conditions of a magical tree.

By Aditya Gurjar(AU23B1004)

Under The Guidance Of Dr. Dinesh Patel Sir

Problem Statement

Condition 1

The sum of the values of identical child nodes in both trees must be the same.

Condition 2

The difference between each child node's value and its parent node's value must be the same for both trees.

The assignment involves constructing two full binary trees, applying a traversal technique, and verifying the conditions.

Implementation Steps

Creation of Full Binary Trees

The program prompts the user to enter values to construct two binary trees with identical structures.

If the user enters -1, it indicates no node is present at that position.

Traversal of the Trees

Post-order traversal is used to process nodes recursively (left subtree, right subtree, then root).

Sum Calculation

The sum of identical child nodes is computed for both trees and compared.

Difference Calculation

The absolute difference between the left and right child nodes is calculated.

The difference between this value and the parent node is then computed.

Verification of Magical Tree Conditions

If both conditions are met for all nodes up to the root, the trees are magical.

Otherwise, they are not.

Code Implementation

Node Structure

A Node structure to represent tree nodes.

createTree() Function

For user input-based tree construction.

postOrderTraversal() Function

For tree traversal.

areMagical() Function

To validate the magical tree conditions.

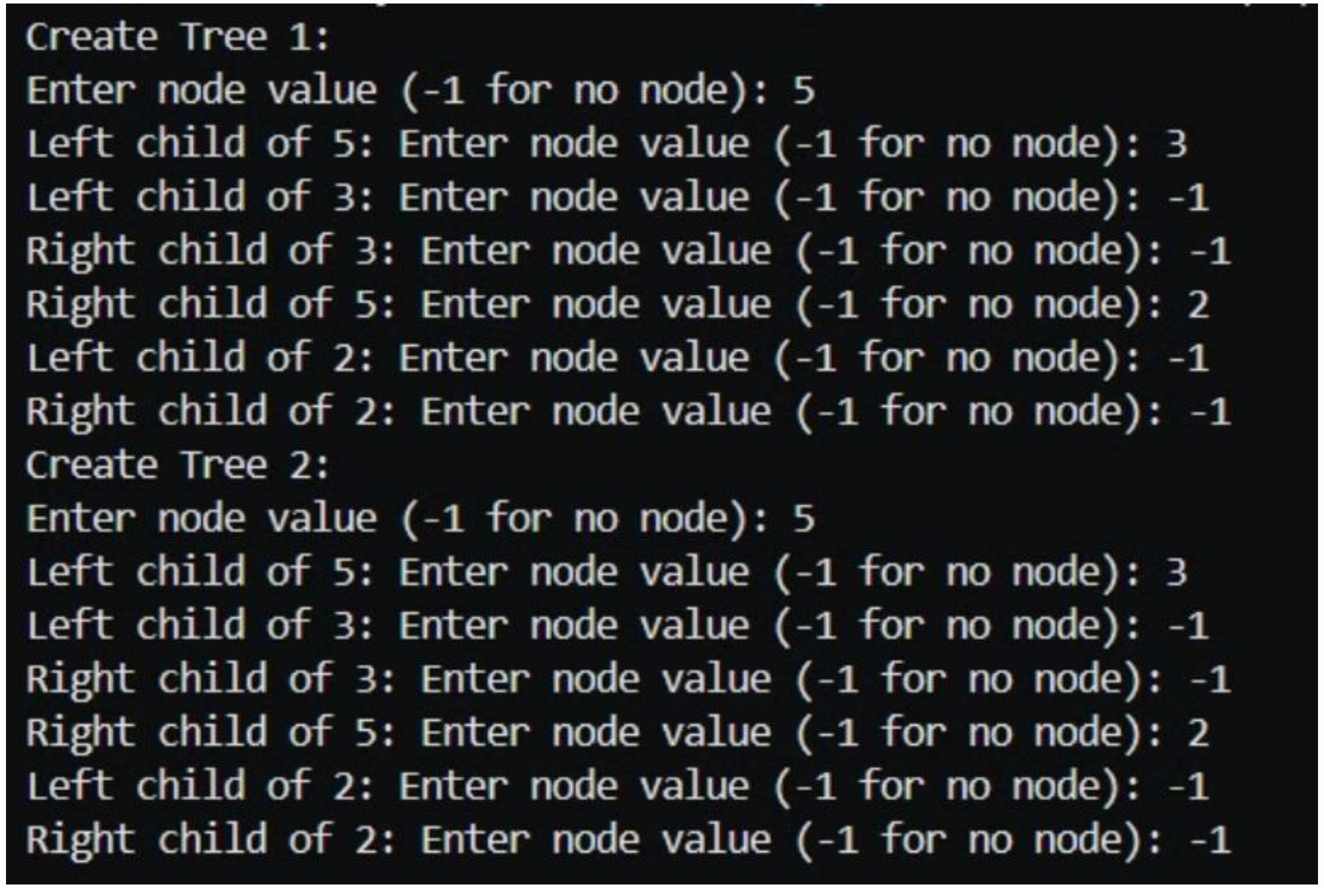
main() Function

That integrates all components and executes the program.

The program is implemented in C++ and includes these key components.

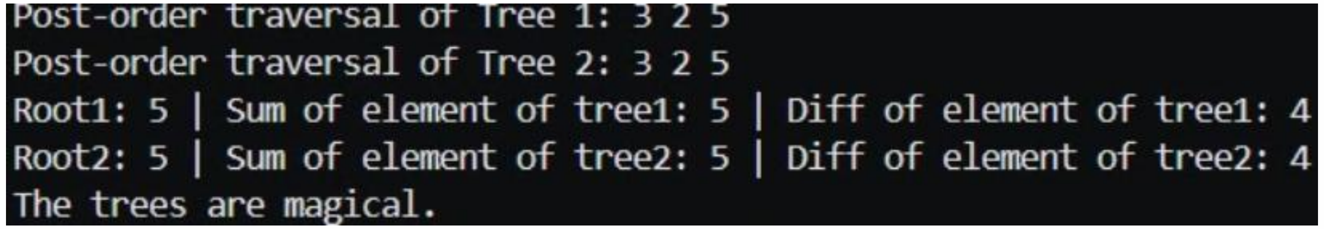
Sample Execution

Input:



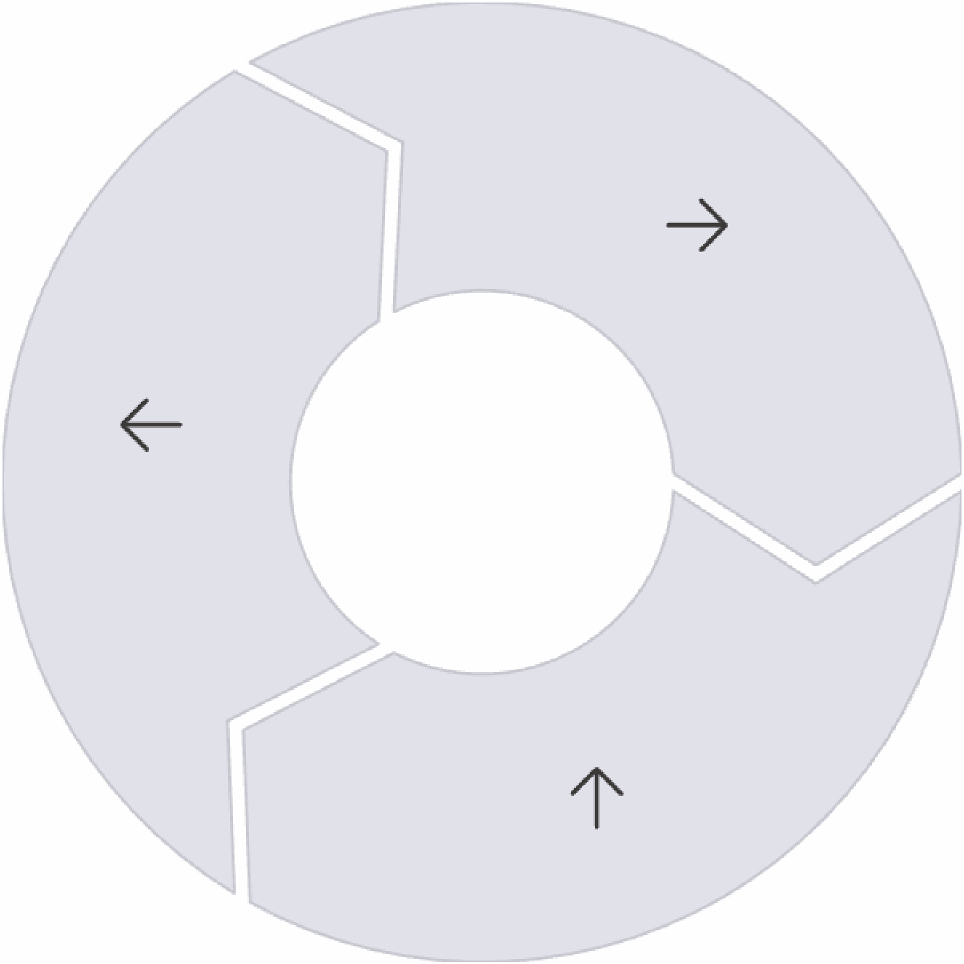
```
Create Tree 1:
Enter node value (-1 for no node): 5
Left child of 5: Enter node value (-1 for no node): 3
Left child of 3: Enter node value (-1 for no node): -1
Right child of 3: Enter node value (-1 for no node): -1
Right child of 5: Enter node value (-1 for no node): 2
Left child of 2: Enter node value (-1 for no node): -1
Right child of 2: Enter node value (-1 for no node): -1
Create Tree 2:
Enter node value (-1 for no node): 5
Left child of 5: Enter node value (-1 for no node): 3
Left child of 3: Enter node value (-1 for no node): -1
Right child of 3: Enter node value (-1 for no node): -1
Right child of 5: Enter node value (-1 for no node): 2
Left child of 2: Enter node value (-1 for no node): -1
Right child of 2: Enter node value (-1 for no node): -1
```

Output:



```
Post-order traversal of Tree 1: 3 2 5
Post-order traversal of Tree 2: 3 2 5
Root1: 5 | Sum of element of tree1: 5 | Diff of element of tree1: 4
Root2: 5 | Sum of element of tree2: 5 | Diff of element of tree2: 4
The trees are magical.
```

Traversal Technique



Process Left Subtree

First visit all nodes in the left subtree recursively

Process Right Subtree

Then visit all nodes in the right subtree recursively

Process Root

Finally process the root node after both subtrees

.

Conclusion

Successful Implementation

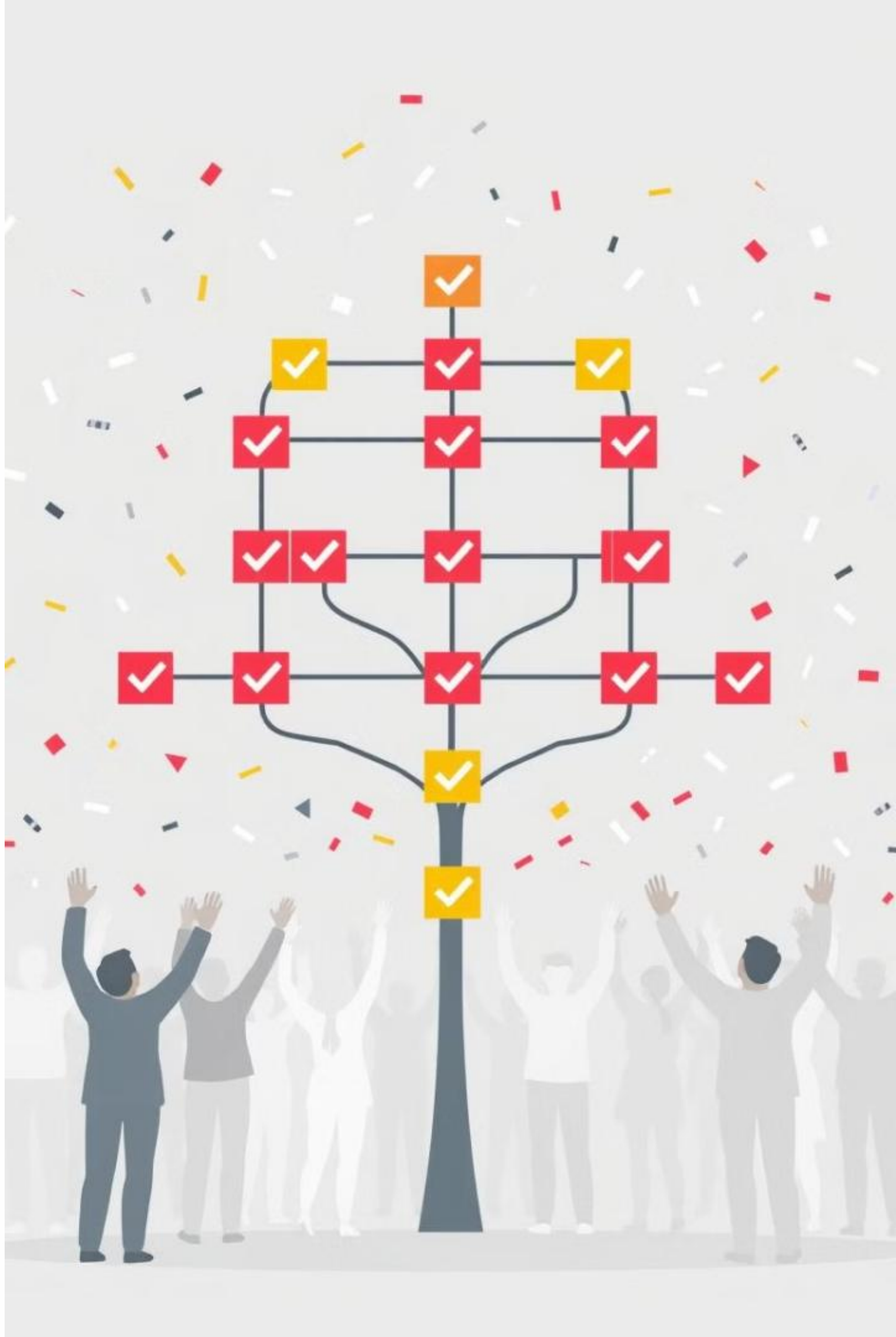
The assignment successfully implements the concept of full binary trees and verifies whether two trees satisfy the magical tree conditions.

Non-Linear Data Structures

The project demonstrates understanding of non-linear data structures using full binary trees.

Magical Tree Verification

The implementation correctly determines whether two given trees satisfy the conditions of a magical tree.



That's All

Thank You!!!