**Minor Project Report**

**Title: Check if Tree is Balanced**

**Index**

1. Introduction
2. Objective
3. Technologies Used
4. Working Principle
5. C++ Code (Core Logic)
6. UI Version – Simple Interface
7. Output Screenshots
8. Conclusion
9. GitHub Project Link

**1. Introduction**

This project focuses on binary trees, particularly on determining whether a binary tree is height-balanced. A binary tree is considered balanced when, for every node, the height difference between its left and right subtrees is at most one.

While working on this project, I initially found recursion a bit tricky to visualize, but after implementing the logic step-by-step, it became much clearer.

**2. Objective**

The main objective of this project was to:

* Take a binary tree input from the user
* Check whether the tree is height-balanced
* Display the result on both a C++ console and a web interface

This helped me improve my logic building and frontend integration skills.

**3. Technologies Used**

* C++ (for the core logic)
* HTML, CSS, JavaScript (for the web UI)
* Browser (to run and test the UI)

**4. Working Principle**

Here’s how the program works:

* The user inputs tree nodes in **preorder** format, where -1 is used to denote NULL nodes.
* The tree is constructed recursively.
* A recursive function calculates the height of both left and right subtrees for each node.
* If the height difference exceeds one for any node, the tree is considered unbalanced.
* In the web UI, users can test the logic by entering their own tree structure.

At first, I had tried checking the height separately, but combining height calculation with balance checking made the code more efficient.

**5. C++ Code (Core Logic)**

#include <iostream>

#include <algorithm>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = NULL;

}

};

Node\* buildTree() {

int val;

cin >> val;

if (val == -1) return NULL;

Node\* node = new Node(val);

node->left = buildTree();

node->right = buildTree();

return node;

}

int balancedHeight(Node\* root) {

if (!root) return 0;

int leftH = balancedHeight(root->left);

if (leftH == -1) return -1;

int rightH = balancedHeight(root->right);

if (rightH == -1) return -1;

if (abs(leftH - rightH) > 1) return -1;

return max(leftH, rightH) + 1;

}

bool isBalanced(Node\* root) {

return balancedHeight(root) != -1;

}

int main() {

cout << "Enter tree nodes (preorder, use -1 for NULL):\n";

Node\* root = buildTree();

if (isBalanced(root)) {

cout << "Balanced\n";

} else {

cout << "Not Balanced\n";

}

return 0;

}

**6. UI Version – Simple Interface**

For the frontend, I designed a lightweight web interface that:

* Takes tree input in preorder format via a text input field
* Has a “Check Balance” button to trigger the check
* Displays results using simple human-readable messages
* Uses a clean dark theme layout with subtle styling

**Key Features:**

* Minimal design
* Basic animations for interaction
* Message delay to simulate a more natural response

**Tools Used:**

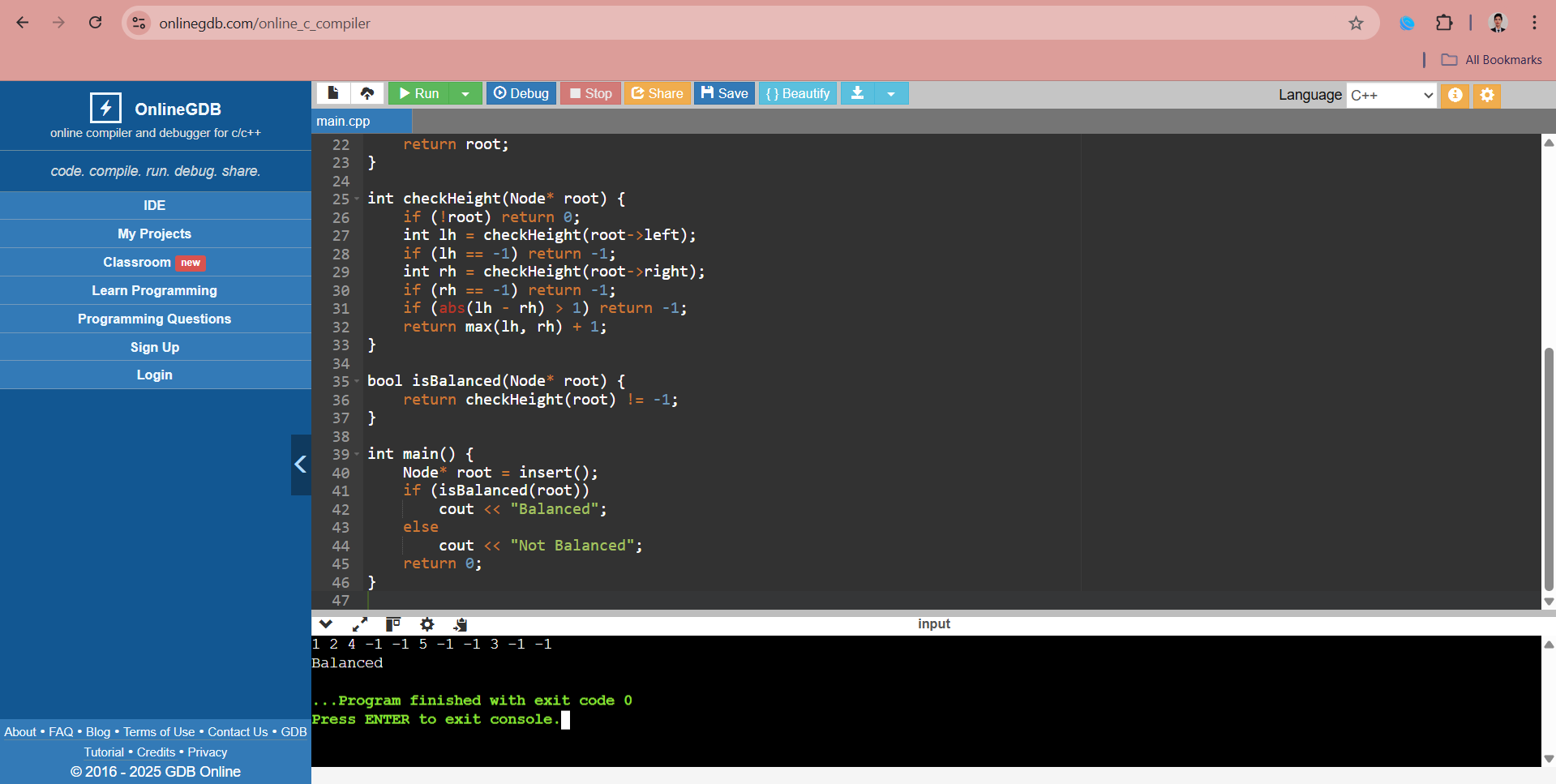
* HTML
* CSS
* JavaScript (Vanilla)

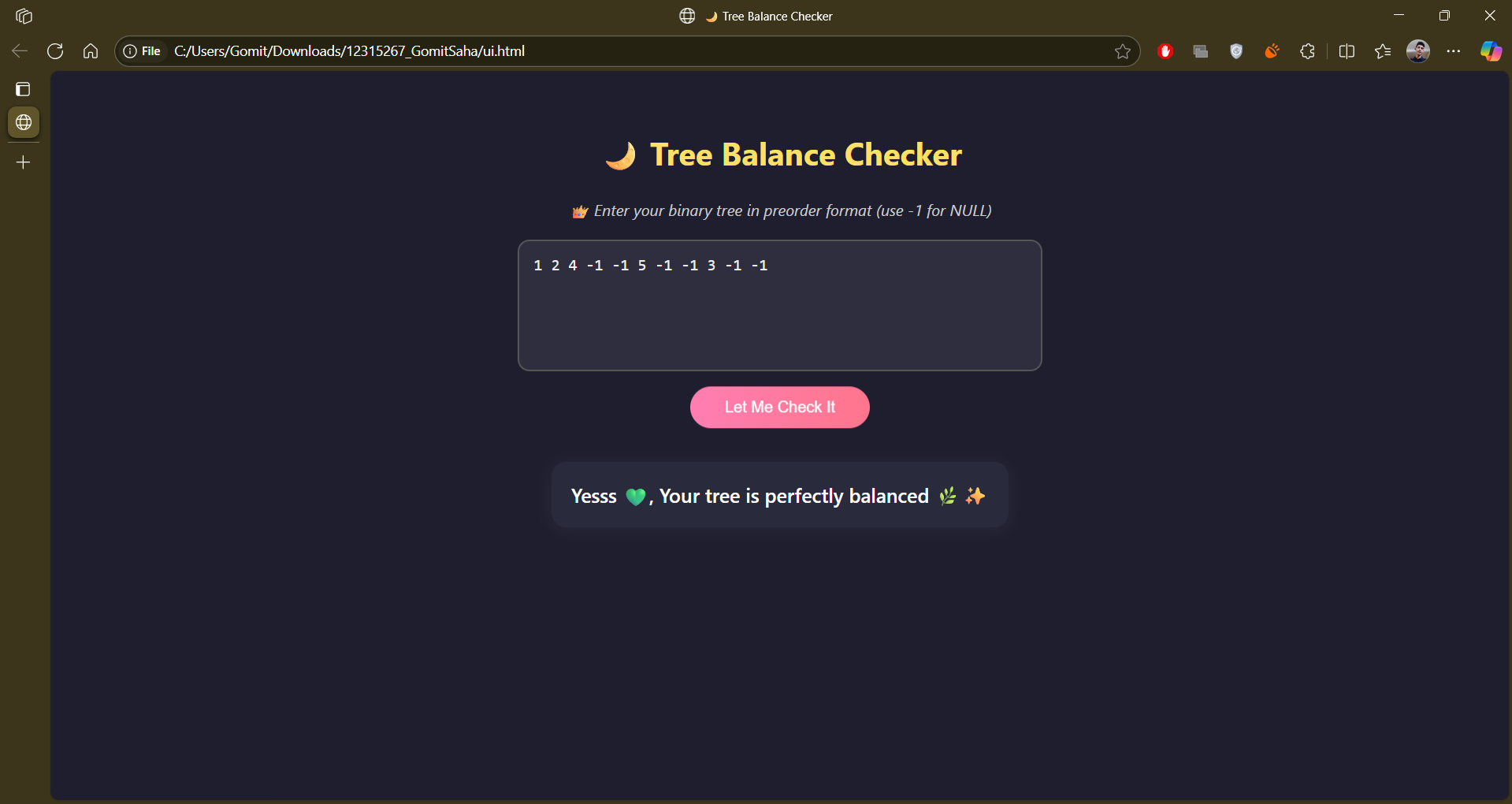
**User Experience:**

Though basic, the UI helps in testing tree inputs quickly and provides a simple, engaging interface to understand the concept visually.

**7. Output Screenshots**

* Console output from C++
* Screenshot of input/output from the UI





**8. Conclusion**

This minor project helped me develop a deeper understanding of:

* Recursive tree construction
* Height and balance checking logic
* Writing clean and efficient C++ code
* Connecting backend logic with frontend UI using JavaScript

It was a valuable learning experience, and it boosted my confidence in both data structures and simple web development.

**9. GitHub Project Link**

[*https://github.com/Gomit-Dev/Check-if-Tree-is-Balanced*](https://github.com/Gomit-Dev/Check-if-Tree-is-Balanced)