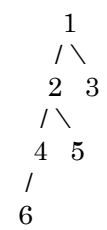
Diameter in C++

```
#include <iostream>
#include <algorithm> // For std::max
using namespace std;

// Definition of the Node class
class Node {
public:
    int key;
    Node* left;
    Node* right;

    Node(int item) {
        key = item;
        left = right = nullptr;
    }
};

// Function prototype for height
int height(Node* node, int* diameter);

// Function to calculate diameter of binary tree
int diameterOfBinaryTree(Node* root) {
    int diameter = 0;
    height(root, &diameter);
    return diameter;
}

// Helper function to calculate height and
update diameter
int height(Node* node, int* diameter) {
    if (node == nullptr) {
        return 0;
    }

    int leftHeight = height(node->left,
diameter);
    int rightHeight = height(node->right,
diameter);

    *diameter = max(*diameter, leftHeight +
rightHeight);

    return 1 + max(leftHeight, rightHeight);
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->left->left->left = new Node(6);

    int dia = diameterOfBinaryTree(root);
    cout << "Diameter of the binary tree: " <<
dia << endl;

    return 0;
}
```

Tree Structure

Based on your construction, the tree looks like this:



🔍 What Is *Diameter*?

The **diameter** is the **length of the longest path** between any two nodes in the tree (measured by number of edges, not nodes).
This path **does not necessarily pass through the root**.

🧠 Core Logic Summary

- For each node:
  - Compute leftHeight and rightHeight.
  - Update diameter = max(diameter, leftHeight + rightHeight).
- Height is returned as 1 + max(leftHeight, rightHeight).

📋 Dry Run Table

Node	Left Height	Right Height	Local Diameter (L + R)	Max Diameter So Far	Returned Height
6	0	0	0	0	1
4	1	0	1	1	2
5	0	0	0	1	1
2	2	1	3	✅ 3	3
3	0	0	0	3	1
1	3	1	4	✅ 4	4

✅ Final Output

Diameter of the binary tree: 4

Diameter of the binary tree: 4