Top View in C++

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <map>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) {
        val = x;
        left = nullptr;
        right = nullptr;
    }
};

// Function to compute the top view of a binary tree
vector<int> topView(TreeNode* root) {
    vector<int> topViewNodes;
    if (!root) {
        return topViewNodes;
    }

    map<int, int> hdMap; // Horizontal Distance Map (hd -> node value)
    queue<pair<TreeNode*, int>> q; // Queue to store nodes and their horizontal distance

    q.push({root, 0}); // Start with the root node at horizontal distance 0

    while (!q.empty()) {
        TreeNode* node = q.front().first;
        int hd = q.front().second;
        q.pop();

        // If this horizontal distance is not already in the map, add the node value
        if (hdMap.find(hd) == hdMap.end()) {
            hdMap[hd] = node->val;
        }

        // Enqueue left and right children with updated horizontal distances
        if (node->left) {
            q.push({node->left, hd - 1});
        }

        if (node->right) {
            q.push({node->right, hd + 1});
        }
    }

    // Extract values from the map in order of horizontal distance
    for (const auto& pair : hdMap) {
        topViewNodes.push_back(pair.second);
    }
```

Constructed Binary Tree:

```
    1
   / \
  2   3
   \
    4
     \
      5
       \
        6
```

📘 Step-by-Step Traversal Table (Level Order with HD)

We'll perform a BFS traversal and track each node with its **Horizontal Distance (HD)** from root.

Step	Queue Content	Popped Node	HD	hdMap Before	hdMap After
1	(1, 0)	1	0	{}	{0: 1}
2	(2, -1), (3, 1)	2	-1	{0: 1}	{-1: 2, 0: 1}
3	(3, 1), (4, 0)	3	1	{-1: 2, 0: 1}	{-1: 2, 0: 1, 1: 3}
4	(4, 0), (5, 1)	4	0	already filled	(no change)
5	(5, 1), (6, 2)	5	1	already filled	(no change)
6	(6, 2)	6	2	{-1: 2, 0: 1, 1: 3}	{… , 2: 6}

🟢 **Final Map (hdMap) Sorted by HD:**

-1 → 2
 0 → 1
 1 → 3
 2 → 6

✅ **Output (Top View):**

2 1 3 6

```
    return topViewNodes;
}

// Utility function to create a new node
TreeNode* newNode(int key) {
    TreeNode* node = new TreeNode(key);
    return node;
}

int main() {
    // Constructing the binary tree
    TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->right = newNode(4);
    root->left->right->right = newNode(5);
    root->left->right->right->right = newNode(6);

    // Get the top view of the binary tree
    vector<int> result = topView(root);

    // Print the top view of the binary tree
    cout << "Top view of the binary tree:" << endl;
    for (int nodeValue : result) {
        cout << nodeValue << " ";
    }
    cout << endl;

    // Clean up memory (optional in this example)
    // You may need to delete nodes if not using smart
pointers
    return 0;
}
```

```
Top view of the binary tree:
2 1 3 6
```