|  |  |
| --- | --- |
| **LCA in C++** | |
| #include <iostream>  using namespace std;  // Definition of a binary tree node  struct Node {  int data;  Node \*left, \*right;  Node(int item) {  data = item;  left = nullptr;  right = nullptr;  }  };  // Function to find the Lowest Common Ancestor (LCA) of two nodes  Node\* getLCA(Node\* root, int a, int b) {  if (root == nullptr) {  return nullptr;  }  if (root->data == a || root->data == b) {  return root;  }  Node\* lca1 = getLCA(root->left, a, b);  Node\* lca2 = getLCA(root->right, a, b);  if (lca1 != nullptr && lca2 != nullptr) {  return root;  }  if (lca1 != nullptr) {  return lca1;  } else {  return lca2;  }  }  // Function to create a binary tree and find LCA  int main() {  // Hardcoded tree construction  Node\* root = new Node(6);  root->left = new Node(3);  root->right = new Node(8);  root->right->left = new Node(7);  root->right->right = new Node(9);  // Find LCA of nodes 3 and 7  Node\* lcaNode = getLCA(root, 3, 7);  cout << "Lowest Common Ancestor of 3 and 7 is: " << lcaNode->data << endl;  // Clean up dynamically allocated memory  delete root->right->right;  delete root->right->left;  delete root->left;  delete root;  return 0;  } | Tree Structure: 6  / \  3 8  / \  7 9  You're finding the **LCA of 3 and 7**. 🔍 Dry Run of getLCA(root, 3, 7):  | **Function Call** | **Returns** | **Reason** | | --- | --- | --- | | getLCA(6, 3, 7) | → 6 | Found 3 in left subtree, 7 in right subtree → current is LCA | | ├── getLCA(3, 3, 7) | → 3 | root->data == a (found node 3) | | └── getLCA(8, 3, 7) | → 7 | found 7 in left subtree, right subtree (9) doesn't contain target | | ├── getLCA(7, 3, 7) | → 7 | root->data == b (found node 7) | | └── getLCA(9, 3, 7) | → nullptr | no match |  ✅ Output: Lowest Common Ancestor of 3 and 7 is: 6 |
| Lowest Common Ancestor of 3 and 7 is: 6 | |