|  |  |
| --- | --- |
| **All single child parent in C++** | |
| #include <iostream>  #include <vector>  using namespace std;  // Definition of a Node in the Binary Tree  struct Node {  int val;  Node\* left;  Node\* right;  Node(int item) {  val = item;  left = nullptr;  right = nullptr;  }  };  // Function to find all nodes with exactly one child  void exactlyOneChild(Node\* root, vector<int>& ans) {  if (root == nullptr || (root->left == nullptr && root->right == nullptr)) {  return;  }    if (root->left == nullptr || root->right == nullptr) {  ans.push\_back(root->val);  }    exactlyOneChild(root->left, ans);  exactlyOneChild(root->right, ans);  }  // Wrapper function for exactlyOneChild  vector<int> exactlyOneChild(Node\* root) {  vector<int> res;  exactlyOneChild(root, res);  return res;  }  int main() {  // Constructing the example binary tree  Node\* root = new Node(1);  root->left = new Node(2);  root->right = new Node(3);  root->left->left = new Node(4);  root->left->left->left = new Node(5);  // Finding nodes with exactly one child  vector<int> ans = exactlyOneChild(root);  // Printing the result  cout << "Nodes with exactly one child: ";  for (int num : ans) {  cout << num << " ";  }  cout << endl;  return 0;  } | Tree Structure: 1  / \  2 3  /  4  /  5 🔍 Nodes with Exactly One Child We traverse and look for nodes that have **only one** non-null child:   | **Node** | **Left Child** | **Right Child** | **Exactly One Child?** | **Added to ans?** | | --- | --- | --- | --- | --- | | 1 | 2 | 3 | ❌ (has both) | ❌ | | 2 | 4 | nullptr | ✅ | ✅ → 2 | | 4 | 5 | nullptr | ✅ | ✅ → 4 | | 5 | nullptr | nullptr | ❌ (no children) | ❌ | | 3 | nullptr | nullptr | ❌ (no children) | ❌ |  ✅ Final Output: Nodes with exactly one child: 2 4 |
| Nodes with exactly one child: 2 4 | |