|  |  |
| --- | --- |
| **Elements in Range in C++** | |
| #include <iostream>  using namespace std;  class ElementsinRange  {  public:  struct Node  {  int key;  Node \*left;  Node \*right;  Node(int item)  {  key = item;  left = nullptr;  right = nullptr;  }  };  static void elementsInRangeK1K2(Node \*root, int k1, int k2)  {  if (root == nullptr)  {  return;  }  if (root->key >= k1 && root->key <= k2)  {  cout << root->key << " ";  }  if (root->key > k1)  {  elementsInRangeK1K2(root->left, k1, k2);  }  if (root->key < k2)  {  elementsInRangeK1K2(root->right, k1, k2);  }  }  };  int main()  {  ElementsinRange::Node \*root = new ElementsinRange::Node(6);  root->left = new ElementsinRange::Node(3);  root->right = new ElementsinRange::Node(8);  root->right->left = new ElementsinRange::Node(7);  root->right->right = new ElementsinRange::Node(9);  cout << "Elements in range [5, 8]: ";  ElementsinRange::elementsInRangeK1K2(root, 5, 8);  cout << endl;  return 0;  } | **BST Structure:**  6  / \  3 8  / \  7 9  **🔍 Traversal Logic:**  The function recursively traverses only relevant subtrees:   * If root->key >= k1, check left subtree. * If root->key <= k2, check right subtree. * If key is within range, print it.   **🧾 Dry Run Table:**   | **Function Call** | **root->key** | **Action Taken** | **Output** | | --- | --- | --- | --- | | elementsInRangeK1K2(6, 5, 8) | 6 | In range → print 6, go left and right | 6 | | ├─ elementsInRangeK1K2(3, 5, 8) | 3 | Not in range, go right skipped | - | | └─ elementsInRangeK1K2(8, 5, 8) | 8 | In range → print 8, go left | 8 | | └─ elementsInRangeK1K2(7, 5, 8) | 7 | In range → print 7 | 7 |   **🖨 Final Output:**  Elements in range [5, 8]: 6 8 7 |
| Elements in range [5, 8]: 6 8 7 | |