|  |  |
| --- | --- |
| **BST in C++** | |
| #include <bits/stdc++.h>  using namespace std;  struct Node {  int key, lcount;  Node \*left, \*right;  Node(int item) {  key = item;  left = nullptr;  right = nullptr;  lcount = 0;  }  };  Node\* insertRec(Node\* root, int key) {  if (root == nullptr) {  return new Node(key);  }  if (key < root->key) {  root->left = insertRec(root->left, key);  root->lcount++;  }  if (key > root->key) {  root->right = insertRec(root->right, key);  }  return root;  }  void inorder(Node\* root) {  if (root == nullptr) {  return;  }  inorder(root->left);  cout << root->key << " ";  inorder(root->right);  }  bool searchRec(Node\* root, int x) {  if (root == nullptr) {  return false;  }  if (x == root->key) {  return true;  }  if (x < root->key) {  return searchRec(root->left, x);  }  return searchRec(root->right, x);  }  bool searchIterative(Node\* root, int x) {  Node\* curr = root;  while (curr != nullptr) {  if (x == curr->key) {  return true;  }  if (x < curr->key) {  curr = curr->left;  }  else {  curr = curr->right;  }  }  return false;  }  Node\* kthSmallestNode(Node\* root, int k) {  if (root == nullptr) {  return nullptr;  }  int count = root->lcount + 1;  if (count == k) {  return root;  }  if (count > k) {  return kthSmallestNode(root->left, k);  }  return kthSmallestNode(root->right, k - count);  }  int kthSmallest(Node\* root, int k) {  Node\* result = kthSmallestNode(root, k);  return result ? result->key : -1;  }  Node\* floor(Node\* root, int x) {  Node\* res = nullptr;  while (root != nullptr) {  if (x == root->key) {  return root;  }  else if (x < root->key) {  root = root->left;  }  else {  res = root;  root = root->right;  }  }  return res;  }  Node\* ceiling(Node\* root, int x) {  Node\* res = nullptr;  while (root != nullptr) {  if (x == root->key) {  return root;  }  else if (x < root->key) {  res = root;  root = root->left;  }  else {  root = root->right;  }  }  return res;  }  int floorValue(Node\* root, int x) {  Node\* result = floor(root, x);  return result ? result->key : -1;  }  int ceilingValue(Node\* root, int x) {  Node\* result = ceiling(root, x);  return result ? result->key : -1;  }  bool isBST(Node\* root, int min, int max) {  if (root == nullptr) {  return true;  }  return (root->key > min && root->key < max &&  isBST(root->left, min, root->key) &&  isBST(root->right, root->key, max));  }  bool isPairSum(Node\* root, int sum, unordered\_set<int>& s) {  if (root == nullptr)  return false;  if (isPairSum(root->left, sum, s)) {  return true;  }  if (s.find(sum - root->key) != s.end()) {  return true;  }  s.insert(root->key);  return isPairSum(root->right, sum, s);  }  bool findPairWithSum(Node\* root, int sum) {  unordered\_set<int> s;  return isPairSum(root, sum, s);  }  int main() {  Node\* root = nullptr;    int data[] = {50, 30, 20, 40, 70, 60, 80};  for (int i : data) {  root = insertRec(root, i);  }  cout << "Inorder traversal: ";  inorder(root);  cout << "\n";  cout << "80 present? " << (searchRec(root, 80) ? "Yes" : "No") << "\n";  cout << "90 present? " << (searchRec(root, 90) ? "Yes" : "No") << "\n";  cout << "81 present? " << (searchIterative(root, 81) ? "Yes" : "No") << "\n";  cout << "Is BST? " << (isBST(root, INT\_MIN, INT\_MAX) ? "Yes" : "No") << "\n";  cout << "Floor of 58? " << floorValue(root, 58) << "\n";  cout << "Ceiling of 58? " << ceilingValue(root, 58) << "\n";  cout << "4th smallest element? " << kthSmallest(root, 4) << "\n";  cout << "Pair with sum 70? " << (findPairWithSum(root, 70) ? "Yes" : "No") << "\n";  return 0;  } | Absolutely! Let's do a **dry run using a detailed table** for your given code. We'll use the input:  int data[] = {50, 30, 20, 40, 70, 60, 80};  We'll perform a **step-by-step dry run** showing:   1. **Insertion** 2. **lcount updates** 3. **Inorder Traversal** 4. **kthSmallest(4) logic** 5. **floor(58) and ceiling(58)** 6. **findPairWithSum(70)**  🧩 1. Insertion and lcount Updates  | **Step** | **Inserted Value** | **Tree Structure After Insert** | **Updated lcount** | | --- | --- | --- | --- | | 1 | 50 | 50 | - | | 2 | 30 | 50 ← 30 (left) | 50.lcount = 1 | | 3 | 20 | 50 ← 30 ← 20 (left-left) | 30.lcount = 1 50.lcount = 2 | | 4 | 40 | 50 ← 30 → 40 (right) | 50.lcount = 2 | | 5 | 70 | 50 → 70 | - | | 6 | 60 | 50 → 70 ← 60 | 70.lcount = 1 | | 7 | 80 | 50 → 70 → 80 | - |  🌲 Final BST Structure 50  / \  30 70  / \ / \  20 40 60 80 🔁 2. Inorder Traversal (inorder())  | **Step** | **Visited Node** | **Output So Far** | | --- | --- | --- | | 1 | 20 | 20 | | 2 | 30 | 20 30 | | 3 | 40 | 20 30 40 | | 4 | 50 | 20 30 40 50 | | 5 | 60 | 20 30 40 50 60 | | 6 | 70 | 20 30 40 50 60 70 | | 7 | 80 | 20 30 40 50 60 70 80 |  🎯 3. kthSmallest(root, 4) Goal: Find the 4th smallest element   | **Node** | **lcount** | **Count (lcount + 1)** | **k** | **Decision** | | --- | --- | --- | --- | --- | | 50 | 2 | 3 | 4 | Go to right with k = 1 | | 70 | 1 | 2 | 1 | Go to left with k = 1 | | 60 | 0 | 1 | 1 | **Match found: return 60** ❌ |   Wait! That's not correct.  Hold up! Actually, **lcount of 50 is 2**, so:   * Elements smaller than 50 = 2 (from its left) * k = 4: so we're looking for the 4th smallest   Steps:   * 50 has 2 nodes on left → total = 3 including itself. * So **4th smallest** must be in **right subtree with k = 4 - 3 = 1** * Move to 70 (right)   + 70.lcount = 1 ⇒ count = 2   + k = 1 < count ⇒ Go to left subtree with k = 1   + Left of 70 = 60, lcount = 0 → count = 1 ⇒ Found!   ✅ **Answer: 4th smallest = 60**  Oops! Wait—this reveals a mistake! lcount of root is being updated incorrectly in your code!  Let's correct the dry run:  **Correct tree lcount:**   | **Node** | **lcount** | | --- | --- | | 50 | 2 | | 30 | 1 | | 70 | 1 |  * So root's left subtree (30) has 2 nodes (20, 40) * But root's lcount = **number of nodes in left subtree**: that's 3 (20, 30, 40) → so it should be **3**!   So in your code, lcount is **not correctly reflecting total left-subtree size**, just how many **insertions to left** occurred.  So fix that if needed! Otherwise, in this logic: ✅ kthSmallest(root, 4) returns **50** 🧮 4. Floor and Ceiling of 58  | **Traversal Path** | **Comparison** | **Action** | **Floor** | **Ceiling** | | --- | --- | --- | --- | --- | | Start @ 50 | 58 > 50 | go right | 50 | - | | Go to 70 | 58 < 70 | go left | 50 | 70 | | Go to 60 | 58 < 60 | go left | 50 | 60 | | Left of 60 = ∅ | End |  | 50 | 60 |   ✅ Floor = 50, Ceiling = 60 🧮 5. Pair Sum = 70 (using unordered\_set) Traversal happens **inorder**. Let’s walk through:   | **Current Node** | **Needed = 70 - x** | **Set (s)** | **Found Pair?** | | --- | --- | --- | --- | | 20 | 50 | {20} | No | | 30 | 40 | {20, 30} | No | | 40 | 30 | {20, 30, 40} | ✅ Yes |   ✅ Found 40 + 30 = 70 ✅ Summary Table of Results  | **Task** | **Output** | | --- | --- | | Inorder Traversal | 20 30 40 50 60 70 80 | | Search 80 | Yes | | Search 90 | No | | Search 81 | No | | Is BST | Yes | | Floor of 58 | 50 | | Ceiling of 58 | 60 | | 4th Smallest Element | 50 | | Pair with Sum = 70 | Yes (40+30) | |
| Inorder traversal: 20 30 40 50 60 70 80  80 present? Yes  90 present? No  81 present? No  Is BST? Yes  Floor of 58? 50  Ceiling of 58? 60  4th smallest element? 50  Pair with sum 70? Yes | |