```
1 #include <iostream>
 2
 3 class TreeNode {
 4 public:
        int key;
 6
       TreeNode* left;
7
       TreeNode* right;
 8
       TreeNode(int value) : key(value), left(nullptr), right(nullptr) {}
9
10 };
11
12 class BinaryTree {
13 private:
14
       TreeNode* root;
15
16
       TreeNode* insert(TreeNode* root, int key) {
17
            if (root == nullptr) {
18
                return new TreeNode(key);
19
            }
20
            if (key < root->key) {
21
22
                root->left = insert(root->left, key);
23
            } else if (key > root->key) {
24
                root->right = insert(root->right, key);
25
            }
26
27
           return root;
28
       }
29
30
       TreeNode* deleteNode(TreeNode* root, int key) {
31
            if (root == nullptr) {
32
                return root;
33
            }
34
35
            if (key < root->key) {
36
                root->left = deleteNode(root->left, key);
37
            } else if (key > root->key) {
38
                root->right = deleteNode(root->right, key);
            } else {
39
40
                if (root->left == nullptr) {
41
                    TreeNode* temp = root->right;
42
                    delete root;
43
                    return temp;
44
                } else if (root->right == nullptr) {
45
                    TreeNode* temp = root->left;
46
                    delete root;
47
                    return temp;
48
                }
49
```

```
...ive\Desktop\DSA Lab\Trees\Binary Tree\binarytree2.cpp
```

```
50
                root->key = minValueNode(root->right);
51
                root->right = deleteNode(root->right, root->key);
52
            }
53
54
           return root;
        }
55
56
57
        int minValueNode(TreeNode* node) {
58
            TreeNode* current = node;
59
            while (current->left != nullptr) {
60
                current = current->left;
61
62
            return current->key;
63
       }
64
65
       TreeNode* search(TreeNode* root, int key) {
            if (root == nullptr || root->key == key) {
66
67
                return root;
68
            }
69
            if (key < root->key) {
70
71
                return search(root->left, key);
72
            } else {
73
                return search(root->right, key);
74
            }
75
       }
76
77
   public:
        BinaryTree() : root(nullptr) {}
78
79
80
       void insert(int key) {
81
            root = insert(root, key);
82
       }
83
84
       void deleteNode(int key) {
85
            root = deleteNode(root, key);
86
       }
87
88
       bool search(int key) {
            return search(root, key) != nullptr;
89
90
       }
91 };
92
93 int main() {
94
       BinaryTree bt;
95
        int keys[] = {50, 30, 70, 20, 40, 60, 80};
96
97
98
       for (int key : keys) {
```

```
...ive\Desktop\DSA Lab\Trees\Binary Tree\binarytree2.cpp
                                                                                  3
            bt.insert(key);
100
        }
101
        std::cout << "Searching for 30: " << (bt.search(30) ? "Found" : "Not >
102
          Found") << std::endl;</pre>
103
104
        bt.deleteNode(70);
105
        std::cout << "Searching for 70: " << (bt.search(20) ? "Found" : "Not >
106
          Found") << std::endl;</pre>
107
        return 0;
108
109 }
```

110