

```
1  #include <iostream>
2
3  class TreeNode {
4  public:
5      int key;
6      TreeNode* left;
7      TreeNode* right;
8
9      TreeNode(int value) : key(value), left(nullptr), right(nullptr) {}
10 };
11
12 class BinarySearchTree {
13 private:
14     TreeNode* root;
15
16     TreeNode* insert(TreeNode* root, int key) {
17         if (root == nullptr) {
18             return new TreeNode(key);
19         }
20
21         if (key < root->key) {
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);
39         } else {
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         }
```

```
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 void inorderTraversal(TreeNode* root) {
66     if (root != nullptr) {
67         inorderTraversal(root->left);
68         std::cout << root->key << " ";
69         inorderTraversal(root->right);
70     }
71 }
72
73 public:
74     BinarySearchTree() : root(nullptr) {}
75
76     void insert(int key) {
77         root = insert(root, key);
78     }
79
80     void deleteNode(int key) {
81         root = deleteNode(root, key);
82     }
83
84     void inorderTraversal() {
85         inorderTraversal(root);
86         std::cout << std::endl;
87     }
88 };
89
90 int main() {
91     BinarySearchTree bst;
92
93     int keys[] = {50, 30, 70, 20, 40, 60, 80};
94
95     for (int key : keys) {
96         bst.insert(key);
97     }
98 }
```

```
99     std::cout << "In-order Traversal: ";
100     bst.inorderTraversal();
101
102     bst.deleteNode(50);
103     std::cout << "In-order Traversal after deleting : ";
104     bst.inorderTraversal();
105
106     return 0;
107 }
108
```