

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node {
6      string keyword;
7      string meaning;
8      Node* left;
9      Node* right;
10     Node(string k, string m) : keyword(k), meaning(m), left(NULL), right(NULL) {}
11 };
12
13 class BSTDictionary {
14 public:
15     Node* root;
16     BSTDictionary() : root(NULL) {}
17
18     Node* insert(Node* root, string keyword, string meaning) {
19         if (root == NULL) return new Node(keyword, meaning);
20         if (keyword < root->keyword)
21             root->left = insert(root->left, keyword, meaning);
22         else if (keyword > root->keyword)
23             root->right = insert(root->right, keyword, meaning);
24         else
25             root->meaning = meaning; // Update meaning if keyword exists
26         return root;
27     }
28
29     Node* minValueNode(Node* node) {
30         Node* current = node;
31         while (current && current->left != NULL)
32             current = current->left;
33         return current;
34     }
35
36     Node* deleteNode(Node* root, string keyword) {
37         if (root == NULL) return root;
38         if (keyword < root->keyword)
39             root->left = deleteNode(root->left, keyword);
40         else if (keyword > root->keyword)
41             root->right = deleteNode(root->right, keyword);
42         else {
43             if (root->left == NULL) {
44                 Node* temp = root->right;
45                 delete root;
46                 return temp;
47             } else if (root->right == NULL) {
48                 Node* temp = root->left;
49                 delete root;
50                 return temp;
51             }
52             Node* temp = minValueNode(root->right);
53             root->keyword = temp->keyword;
54             root->meaning = temp->meaning;
55             root->right = deleteNode(root->right, temp->keyword);
56         }
57         return root;
58     }
59 }
```

```
60 void inorder(Node* root) {
61     if (root != NULL) {
62         inorder(root->left);
63         cout << root->keyword << " : " << root->meaning << endl;
64         inorder(root->right);
65     }
66 }
67
68 void reverseInorder(Node* root) {
69     if (root != NULL) {
70         reverseInorder(root->right);
71         cout << root->keyword << " : " << root->meaning << endl;
72         reverseInorder(root->left);
73     }
74 }
75
76 int searchComparisons(Node* root, string keyword) {
77     int comparisons = 0;
78     while (root != NULL) {
79         comparisons++;
80         if (keyword == root->keyword)
81             return comparisons;
82         else if (keyword < root->keyword)
83             root = root->left;
84         else
85             root = root->right;
86     }
87     return -1; // Return -1 if keyword is not found
88 }
89 };
90
91 int main() {
92     BSTDictionary dict;
93     int choice;
94     string keyword, meaning;
95
96     do {
97         cout << "\n1. Add keyword\n2. Delete keyword\n3. Update keyword\n4. Display
(Ascending)\n5. Display (Descending)\n6. Search keyword comparisons\n7. Exit\nEnter
choice: ";
98         cin >> choice;
99         switch(choice) {
100             case 1:
101                 cout << "Enter keyword: "; cin >> keyword;
102                 cout << "Enter meaning: "; cin.ignore(); getline(cin, meaning);
103                 dict.root = dict.insert(dict.root, keyword, meaning);
104                 break;
105             case 2:
106                 cout << "Enter keyword to delete: "; cin >> keyword;
107                 dict.root = dict.deleteNode(dict.root, keyword);
108                 break;
109             case 3:
110                 cout << "Enter keyword to update: "; cin >> keyword;
111                 cout << "Enter new meaning: "; cin.ignore(); getline(cin, meaning);
112                 dict.root = dict.insert(dict.root, keyword, meaning);
113                 break;
114             case 4:
115                 dict.inorder(dict.root);
116                 break;
117             case 5:
118                 dict.reverseInorder(dict.root);
```

```
119         break;
120     case 6:
121         cout << "Enter keyword to search: "; cin >> keyword;
122         {
123             int comparisons = dict.searchComparisons(dict.root, keyword);
124             if (comparisons == -1)
125                 cout << "Keyword not found!\n";
126             else
127                 cout << "Comparisons required to search: " << comparisons <<
endl;
128         }
129         break;
130     case 7:
131         cout << "Exiting...\n";
132         break;
133     default:
134         cout << "Invalid choice!\n";
135     }
136 } while(choice != 7);
137
138 return 0;
139 }
140
```