

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Node
6 {
7     string keyword, meaning;
8     Node* left;
9     Node* right;
10    int height;
11 };
12
13 // Function to get the height of a node
14 int getHeight(Node* n)
15 {
16     if (n == NULL)
17         return 0;
18     else
19         return n->height;
20 }
21
22 // Function to get balance factor
23 int getBalanceFactor(Node* n)
24 {
25     if (n == NULL)
26         return 0;
27     else
28         return getHeight(n->left) - getHeight(n->right);
29 }
30
31 // Function to create a new node
32 Node* createNode(string keyword, string meaning)
33 {
34     Node* node = new Node();
35     node->keyword = keyword;
36     node->meaning = meaning;
37     node->left = node->right = NULL;
38     node->height = 1;
39     return node;
40 }
41
42 // Right rotate
43 Node* rightRotate(Node* y)
44 {
45     Node* x = y->left;
46     Node* T2 = x->right;
47     x->right = y;
48     y->left = T2;
49
50     // Update height of y
51     int leftHeight = getHeight(y->left);
52     int rightHeight = getHeight(y->right);
53     if (leftHeight > rightHeight)
54         y->height = leftHeight + 1;
55     else
```

```

56         y->height = rightHeight + 1;
57
58     // Update height of x
59     leftHeight = getHeight(x->left);
60     rightHeight = getHeight(x->right);
61     if (leftHeight > rightHeight)
62         x->height = leftHeight + 1;
63     else
64         x->height = rightHeight + 1;
65
66     return x;
67 }
68
69 // Left rotate
70 Node* leftRotate(Node* x)
71 {
72     Node* y = x->right;
73     Node* T2 = y->left;
74     y->left = x;
75     x->right = T2;
76
77     // Update height of x
78     int leftHeight = getHeight(x->left);
79     int rightHeight = getHeight(x->right);
80     if (leftHeight > rightHeight)
81         x->height = leftHeight + 1;
82     else
83         x->height = rightHeight + 1;
84
85     // Update height of y
86     leftHeight = getHeight(y->left);
87     rightHeight = getHeight(y->right);
88     if (leftHeight > rightHeight)
89         y->height = leftHeight + 1;
90     else
91         y->height = rightHeight + 1;
92
93     return y;
94 }
95
96
97 // Insert a keyword
98 Node* insert(Node* root, string keyword, string meaning)
99 {
100     if (root == NULL)
101         return createNode(keyword, meaning);
102
103     if (keyword < root->keyword)
104         root->left = insert(root->left, keyword, meaning);
105     else if (keyword > root->keyword)
106         root->right = insert(root->right, keyword, meaning);
107     else {
108         root->meaning = meaning; // Update meaning if keyword exists
109         return root;
110     }
111
112     // Update height of root
113     int leftHeight = getHeight(root->left);

```

```

113     int rightHeight = getHeight(root->right);
114     if (leftHeight > rightHeight)
115         root->height = leftHeight + 1;
116     else
117         root->height = rightHeight + 1;
118
119     int balance = getBalanceFactor(root);
120
121     // Perform rotations if needed
122     if (balance > 1 && keyword < root->left->keyword)
123         return rightRotate(root);
124     if (balance < -1 && keyword > root->right->keyword)
125         return leftRotate(root);
126     if (balance > 1 && keyword > root->left->keyword)
127     {
128         root->left = leftRotate(root->left);
129         return rightRotate(root);
130     }
131     if (balance < -1 && keyword < root->right->keyword)
132     {
133         root->right = rightRotate(root->right);
134         return leftRotate(root);
135     }
136
137     return root;
138 }
139
140 // Find the node with minimum value
141 Node* minValueNode(Node* root)
142 {
143     while (root->left)
144         root = root->left;
145     return root;
146 }
147
148 // Delete a keyword
149 Node* deleteNode(Node* root, string keyword)
150 {
151     if (root == NULL)
152         return root;
153
154     if (keyword < root->keyword)
155         root->left = deleteNode(root->left, keyword);
156     else if (keyword > root->keyword)
157         root->right = deleteNode(root->right, keyword);
158     else
159     {
160         if (!root->left || !root->right)
161         {
162             Node* temp = root->left ? root->left : root->right;
163             if (!temp)
164                 temp = root, root = NULL;
165             else
166                 *root = *temp;
167             delete temp;
168         }
169         else

```

```

170
171     {
172         Node* temp = minValueNode(root->right);
173         root->keyword = temp->keyword;
174         root->meaning = temp->meaning;
175         root->right = deleteNode(root->right, temp->keyword);
176     }
177 }
178 if (!root)
179     return root;
180
181 // Update height of root
182 int leftHeight = getHeight(root->left);
183 int rightHeight = getHeight(root->right);
184 if (leftHeight > rightHeight)
185     root->height = leftHeight + 1;
186 else
187     root->height = rightHeight + 1;
188
189 int balance = getBalanceFactor(root);
190
191 if (balance > 1 && getBalanceFactor(root->left) >= 0)
192     return rightRotate(root);
193 if (balance > 1 && getBalanceFactor(root->left) < 0)
194 {
195     root->left = leftRotate(root->left);
196     return rightRotate(root);
197 }
198 if (balance < -1 && getBalanceFactor(root->right) <= 0)
199     return leftRotate(root);
200 if (balance < -1 && getBalanceFactor(root->right) > 0)
201 {
202     root->right = rightRotate(root->right);
203     return leftRotate(root);
204 }
205
206 return root;
207 }
208
209 // Display dictionary in ascending order
210 void inOrder(Node* root)
211 {
212     if (root)
213     {
214         inOrder(root->left);
215         cout << root->keyword << " : " << root->meaning << endl;
216         inOrder(root->right);
217     }
218 }
219
220
221 // Display dictionary in descending order
222 void reverseInOrder(Node* root)
223 {
224     if (root)
225     {
226         reverseInOrder(root->right);
227         cout << root->keyword << " : " << root->meaning << endl;

```

```

227         reverseInOrder(root->left);
228     }
229 }
230
231 // Search for a keyword and count comparisons
232 int search(Node* root, string keyword, int comparisons = 0)
233 {
234     if (root == NULL)
235         return comparisons;
236     comparisons++;
237     if (keyword == root->keyword)
238         return comparisons;
239     else if (keyword < root->keyword)
240         return search(root->left, keyword, comparisons);
241     else
242         return search(root->right, keyword, comparisons);
243 }
244
245 // Update the meaning of a keyword
246 void updateMeaning(Node* root, string keyword, string newMeaning)
247 {
248     if (root == NULL)
249     {
250         cout << "Keyword not found!\n";
251         return;
252     }
253     if (keyword < root->keyword)
254         updateMeaning(root->left, keyword, newMeaning);
255     else if (keyword > root->keyword)
256         updateMeaning(root->right, keyword, newMeaning);
257     else
258         root->meaning = newMeaning;
259 }
260
261 int main()
262 {
263     Node* root = NULL;
264     int choice;
265     string keyword, meaning;
266
267     do {
268         cout << "\nDictionary Operations:\n";
269         cout << "1. Insert\n2. Delete\n3. Update Meaning\n4. Display Ascending\n5.
Display Descending\n6. Search\n7. Exit\n";
270         cout << "Enter your choice: ";
271         cin >> choice;
272
273         switch (choice)
274         {
275             case 1:
276                 cout << "Enter keyword: ";
277                 cin >> keyword;
278                 cout << "Enter meaning: ";
279                 cin.ignore();
280                 getline(cin, meaning);
281                 root = insert(root, keyword, meaning);
282                 break;

```

```
283
284     case 2:
285         cout << "Enter keyword to delete: ";
286         cin >> keyword;
287         root = deleteNode(root, keyword);
288         break;
289
290     case 3:
291         cout << "Enter keyword to update: ";
292         cin >> keyword;
293         cout << "Enter new meaning: ";
294         cin.ignore();
295         getline(cin, meaning);
296         updateMeaning(root, keyword, meaning);
297         break;
298
299     case 4:
300         cout << "\nDictionary (Ascending Order):\n";
301         inOrder(root);
302         break;
303
304     case 5:
305         cout << "\nDictionary (Descending Order):\n";
306         reverseInOrder(root);
307         break;
308
309     case 6:
310         cout << "Enter keyword to search: ";
311         cin >> keyword;
312         cout << "Comparisons required: " << search(root, keyword) << endl;
313         break;
314
315     case 7:
316         cout << "Exiting...\n";
317         break;
318
319     default:
320         cout << "Invalid choice!\n";
321     }
322 }
323 while (choice != 7);
324
325 return 0;
326 }
327 }
```

## OutPut :

D:\SE Computer\LAB CODES\DSA\DSA9.exe

Dictionary Operations:

- 1. Insert
- 2. Delete
- 3. Update Meaning
- 4. Display Ascending
- 5. Display Descending
- 6. Search
- 7. Exit

Enter your choice: 1

Enter keyword: Apple

Enter meaning: A Fruit

Dictionary Operations:

- 1. Insert
- 2. Delete
- 3. Update Meaning
- 4. Display Ascending
- 5. Display Descending
- 6. Search
- 7. Exit

Enter your choice: 1

Enter keyword: Banana

Enter meaning: A Yellow Fruit

Dictionary Operations:

- 1. Insert
- 2. Delete
- 3. Update Meaning
- 4. Display Ascending
- 5. Display Descending
- 6. Search
- 7. Exit

Enter your choice: 1

Enter keyword: Cherry

Enter meaning: A small Fruit

Dictionary Operations:

- 1. Insert
- 2. Delete
- 3. Update Meaning
- 4. Display Ascending
- 5. Display Descending
- 6. Search
- 7. Exit

Enter your choice: 4

Dictionary (Ascending Order):

Apple : A Fruit

Banana : A Yellow Fruit

Select D:\SE Computer\LAB CODES\DSA\DSA9.exe

Apple : A Fruit  
Banana : A Yellow Fruit  
Cherry : A small Fruit

Dictionary Operations:

1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit

Enter your choice: 2

Enter keyword to delete: Banana

Dictionary Operations:

1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit

Enter your choice: 4

Dictionary (Ascending Order):

Apple : A Fruit  
Cherry : A small Fruit

Dictionary Operations:

1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit

Enter your choice: 3

Enter keyword to update: Cherry

Enter new meaning: A small Red fruit

Dictionary Operations:

1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit

Enter your choice: 5

```
■ Select D:\SE Computer\LAB CODES\DSA\DSA9.exe
Dictionary (Descending Order):
Cherry : A small Red fruit
Apple : A Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 6
Enter keyword to search: Banana
Comparisons required: 2

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 7
Exiting...

-----
Process exited after 148.2 seconds with return value 0
Press any key to continue . . .
```