```
#include <iostream>
#include <cstring>
using namespace std;
class Node {
public:
   char word[20], meaning[20];
   Node *left, *right;
    Node(const char w[], const char m[]) {
      strcpy(word, w);
      strcpy(meaning, m);
      left = right = nullptr;
   }
};
class Dictionary {
public:
   Node* root;
   Dictionary() { root = nullptr; }
   Node* insert(Node* root, const char word[], const char meaning[]) {
      if (!root) return new Node(word, meaning);
      if (strcmp(word, root->word) < 0)
         root->left = insert(root->left, word, meaning);
      else if (strcmp(word, root->word) > 0)
         root->right = insert(root->right, word, meaning);
      else
         cout << "Duplicate words are not allowed.\n";
      return root;
   }
   void search(Node* root, const char word[]) {
      if (!root) {
         cout << "Word not found!\n";
         return;
```

```
}
  if (strcmp(word, root->word) == 0) {
        cout << "Word Found!\n";
        cout << "Word: " << root->word << "\nMeaning: " << root->meaning << endl;
    } else if (strcmp(word, root->word) < 0)
        search(root->left, word);
    else
        search(root->right, word);
  }
   Node* findMin(Node* root) {
    while (root && root->left)
        root = root->left;
    return root;
   }
   Node* deleteWord(Node* root, const char word[]) {
    if (!root) {
        cout << "Word not found!\n";
        return root;
    }
   if (strcmp(word, root->word) < 0)
        root->left = deleteWord(root->left, word);
    else if (strcmp(word, root->word) > 0)
        root->right = deleteWord(root->right, word);
    else {
        if (!root->left) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            delete root;
```

```
            return temp;
        }

        Node* temp = findMin(root->right);
        strcpy(root->word, temp->word);
        strcpy(root->meaning, temp->meaning);
        root->right = deleteWord(root->right, temp->word);
    }
    return root;
}

void update(Node* root, const char word[]) {
    if (!root) {
        cout << "Word not found!\n";
        return;
    }

    if (strcmp(word, root->word) == 0) {
        cout << "Enter new meaning: ";
        cin >> root->meaning;
        cout << "Word meaning updated successfully.\n";
    } else if (strcmp(word, root->word) < 0)
        update(root->left, word);
    else
        update(root->right, word);
}

void ascOrder(Node* root) {
    if (root) {
        ascOrder(root->left);
        cout << root->word << " => " << root->meaning << endl;
```

```
            ascOrder(root->right);
        }
    }

    void descOrder(Node* root) {
        if (root) {
            descOrder(root->right);
            cout << root->word << " => " << root->meaning << endl;
            descOrder(root->left);
        }
    }

    int maxComparisons(Node* root) {
        if (!root) return 0;
        int left = maxComparisons(root->left);
        int right = maxComparisons(root->right);
        return max(left, right) + 1;
    }
};

int main() {
    Dictionary dict;
    int choice;
    char word[20], meaning[20];

    while (true) {
        cout << "\nEnter a choice:\n";
        cout << "1) Insert\n2) Search\n3) Delete\n4) Update\n5) Display Ascending\n";
        cout << "6) Display Descending\n7) Max Comparisons for Search\n8) Exit\n";
        cout << "Your choice: ";
        cin >> choice;
```

```
switch (choice) {
    case 1:
        cout << "Enter Word: ";
        cin >> word;
        cout << "Enter Meaning: ";
        cin >> meaning;
        dict.root = dict.insert(dict.root, word, meaning);
        break;

    case 2:
        cout << "Enter Word to be searched: ";
        cin >> word;
        dict.search(dict.root, word);
        break;

    case 3:
        cout << "Enter Word to be deleted: ";
        cin >> word;
        dict.root = dict.deleteWord(dict.root, word);
        break;

    case 4:
        cout << "Enter Word to update meaning: ";
        cin >> word;
        dict.update(dict.root, word);
        break;

    case 5:
        cout << "\nDictionary in Ascending Order:\n";
        dict.ascOrder(dict.root);
```

```
            break;

        case 6:
            cout << "\nDictionary in Descending Order:\n";
            dict.descOrder(dict.root);
            break;

        case 7:
            cout << "Maximum comparisons needed in worst case: "
                 << dict.maxComparisons(dict.root) << endl;
            break;

        case 8:
            cout << "Exiting program.\n";
            return 0;

        default:
            cout << "Invalid choice! Try again.\n";
      }
   }

   return 0;
}
```

Enter a choice:

1) Insert

2) Search

3) Delete

4) Update

5) Display Ascending

6) Display Descending

7) Max Comparisons for Search

8) Exit

Your choice: 1

Enter Word: apple

Enter Meaning: fruit

Your choice: 1

Enter Word: banana

Enter Meaning: yellow

Your choice: 1

Enter Word: cat

Enter Meaning: animal

Your choice: 5

Dictionary in Ascending Order:

apple => fruit

banana => yellow

cat => animal

Your choice: 2

Enter Word to be searched: banana

Word Found!

Word: banana

Meaning: yellow

Your choice: 3

Enter Word to be deleted: banana

Your choice: 2

Enter Word to be searched: banana

Word not found!

Your choice: 4

Enter Word to update meaning: cat

Enter new meaning: pet

Word meaning updated successfully.

Your choice: 6

Dictionary in Descending Order:

cat => pet

apple => fruit

Your choice: 7

Maximum comparisons needed in worst case: 2

Your choice: 8

Exiting program.