```
// Sakshi Yadav (79)
// PR_09
// Title: AVL Tree (Dictionary)

#include <iostream>
#include <string>
using namespace std;

class dict {
private:
   struct Node {
      string s1; // keyword
      string s2; // meaning
      Node* left;
      Node* right;
   };
   Node* root;

public:
   dict() {
      root = NULL;
   }

   void input();
   void create_root(Node* &tree, Node* node);
   void input_display();
   void display(Node* tree);
   void input_remove();
   Node* remove(Node* tree, const string &s);
   Node* findmin(Node* tree);
   void input_find();
   Node* find(Node* tree, const string &s);
   void input_update();
   Node* update(Node* tree, const string &s);
};

void dict::input() {
   Node* node = new Node;
   cout << "\nEnter the keyword:\n";
   cin >> node->s1;
   cout << "Enter the meaning of the keyword:\n";
   cin.ignore();
   getline(cin, node->s2);
   create_root(root, node);
```

```
}

void dict::create_root(Node* &tree, Node* node) {
   if (tree == NULL) {
      tree = node;
      tree->left = NULL;
      tree->right = NULL;
      cout << "\nRoot node created successfully" << endl;
      return;
   }
   if (node->s1 < tree->s1) {
      create_root(tree->left, node);
   } else if (node->s1 > tree->s1) {
      create_root(tree->right, node);
   } else {
      cout << "The word already exists in the dictionary.\n";
      delete node; // Avoid memory leak
   }
}

void dict::input_display() {
   if (root != NULL) {
      cout << "The words entered in the dictionary are:\n\n";
      display(root);
   } else {
      cout << "\nThere are no words in the dictionary.\n";
   }
}

void dict::display(Node* tree) {
   if (tree != NULL) {
      display(tree->left);
      cout << tree->s1 << " = " << tree->s2 << "\n";
      display(tree->right);
   }
}

void dict::input_remove() {
   if (root != NULL) {
      string s;
      cout << "\nEnter a keyword to be deleted:\n";
      cin >> s;
      root = remove(root, s);
   } else {
```

```
        cout << "\nThere are no words in the dictionary.\n";
    }
}

dict::Node* dict::remove(Node* tree, const string &s) {
    if (tree == NULL) {
        cout << "\nWord not found.\n";
        return tree;
    }
    if (s < tree->s1) {
        tree->left = remove(tree->left, s);
    } else if (s > tree->s1) {
        tree->right = remove(tree->right, s);
    } else {
        // Node with only one child or no child
        if (tree->left == NULL) {
            Node* temp = tree->right;
            delete tree;
            return temp;
        } else if (tree->right == NULL) {
            Node* temp = tree->left;
            delete tree;
            return temp;
        }
        // Node with two children: Get the inorder successor
        Node* temp = findmin(tree->right);
        tree->s1 = temp->s1;
        tree->s2 = temp->s2;
        tree->right = remove(tree->right, temp->s1);
    }
    return tree;
}

dict::Node* dict::findmin(Node* tree) {
    while (tree && tree->left != NULL) {
        tree = tree->left;
    }
    return tree;
}

void dict::input_find() {
    if (root != NULL) {
        string s;
        cout << "\nEnter the keyword to be searched:\n";
```

```
        cin >> s;
        find(root, s);
    } else {
        cout << "\nThere are no words in the dictionary.\n";
    }
}

dict::Node* dict::find(Node* tree, const string &s) {
    if (tree == NULL) {
        cout << "\nWord not found.\n";
        return NULL;
    }
    if (s == tree->s1) {
        cout << "\nWord found: " << tree->s1 << " = " << tree->s2 << "\n";
        return tree;
    } else if (s < tree->s1) {
        return find(tree->left, s);
    } else {
        return find(tree->right, s);
    }
}

void dict::input_update() {
    // Update functionality can be implemented here
    cout << "\nUpdate feature not implemented yet.\n";
}

int main() {
    dict d;
    int ch;
    do {
        cout << "\n1. Input\n2. Display\n3. Remove\n4. Find\n5. Update\n6. Exit\n";
        cout << "Enter your choice: ";
        cin >> ch;
        switch (ch) {
            case 1:
                d.input();
                break;
            case 2:
                d.input_display();
                break;
            case 3:
                d.input_remove();
                break;
```

```
            case 4:
                d.input_find();
                break;
            case 5:
                d.input_update();
                break;
            case 6:
                cout << "Exiting...\n";
                break;
            default:
                cout << "\nPlease enter a valid option!\n";
                break;
        }
    } while (ch != 6);
    return 0;
}
```