```
#include <iostream>
#include <string>
using namespace std;
class avl_node{
    string word, meaning;
    avl_node *left;
    avl_node *right;
    friend class avlTree;};
class avlTree{
    avl_node *root;
public:
    avlTree(){
        root = NULL;}
    int height(avl_node *);
    int diff(avl_node *);
    avl_node *ll_rotation(avl_node *);
    avl_node *rr_rotation(avl_node *);
    avl_node *lr_rotation(avl_node *);
    avl_node *rl_rotation(avl_node *);
    avl_node *balance(avl_node *);
    void insert();
    avl_node *insert(avl_node *, avl_node *);
    avl_node *search(string);
    void update(string, string);
    void display();
    void inorder(avl_node *);
    void preorder(avl_node *);
    void postorder(avl_node *);};
int avlTree::height(avl_node *temp){
    int h = 0;
    if (temp != NULL){
        int l_height = height(temp->left);
        int r_height = height(temp->right);
        int max_height = max(l_height, r_height);
        h = max_height + 1;}
    return h;}
int avlTree::diff(avl_node *temp){
    int l_height = height(temp->left);
    int r_height = height(temp->right);
    int b_factor = l_height - r_height;
    return b_factor;}
avl_node *avlTree::ll_rotation(avl_node *parent){
    avl_node *temp = parent->left;
    parent->left = temp->right;
    temp->right = parent;
    return temp;}
avl_node *avlTree::rr_rotation(avl_node *parent){
    avl_node *temp = parent->right;
    parent->right = temp->left;
    temp->left = parent;
    return temp;}
avl_node *avlTree::lr_rotation(avl_node *parent){
    avl_node *temp = parent->left;
    parent->left = rr_rotation(temp);
    return ll_rotation(parent);}
avl_node *avlTree::rl_rotation(avl_node *parent){
    avl_node *temp = parent->right;
    parent->right = ll_rotation(temp);
  return rr_rotation(parent);}
avl_node *avlTree::balance(avl_node *temp){
    int bal_factor = diff(temp);
    if (bal_factor > 1){
        if (diff(temp->left) > 0)
            temp = ll_rotation(temp);
        else
            temp = lr_rotation(temp);}
    else if (bal_factor < -1){
        if (diff(temp->right) > 0)
```

```
            temp = rl_rotation(temp);
        else
            temp = rr_rotation(temp);}
    return temp;}
avl_node *avlTree::insert(avl_node *root, avl_node *temp){
    if (root == NULL){
        root = new avl_node;
        root->word = temp->word;
        root->meaning = temp->meaning;
        root->left = NULL;
        root->right = NULL;
        return root;}
if (temp->word < root->word){
        root->left = insert(root->left, temp);}
    else if (temp->word > root->word){
        root->right = insert(root->right, temp);}
     root = balance(root);
    return root;}
void avlTree::insert(){
    string word, meaning;
    char ch;
    do{
        cout << "Enter word to be inserted: ";
        cin >> word;
        cin.ignore();
        cout << "Enter meaning: ";
        getline(cin, meaning);
        avl_node *temp = new avl_node;
        temp->word = word;
        temp->meaning = meaning;
        temp->left = NULL;
        temp->right = NULL;
        root = insert(root, temp);
        cout << "Word inserted successfully!\n";
        display();
        cout << "Do you want to add more words? (y/n): ";
        cin >> ch;
    } while (ch == 'y' || ch == 'Y');}
avl_node *avlTree::search(string word) {
    int count = 0;
    avl_node *temp = root;
while (temp != NULL) {
        count++;
if (temp->word == word) {
            cout << "Node found after " << count << " comparisons.\n";
            return temp; }
        else if (temp->word > word)
            temp = temp->left;
        else
            temp = temp->right;}}
void avlTree::update(string word, string newMeaning){
    avl_node *result = search(word);
    if (result != NULL){
        result->meaning = newMeaning;
        cout << "Meaning of '" << word << "' updated successfully!\n";}
    else{
        cout << "Word not found. Please insert it first.\n";}}
void avlTree::display(){
    if (root == NULL){
        cout << "Tree is empty\n";
        return;}
    cout << "--------------------------------" << endl;
    cout << "Inorder traversal: " << endl;
    inorder(this->root);
    cout << "--------------------------------" << endl;
    cout << "Preorder traversal: " << endl;
    preorder(this->root);
    cout << "--------------------------------" << endl;
    cout << "Postorder traversal: " << endl;
    postorder(this->root);
    cout << "--------------------------------" << endl;}
void avlTree::inorder(avl_node *node){
    if (node != NULL){
        inorder(node->left);
        cout << node->word << ": " << node->meaning << endl;
        inorder(node->right);}}
void avlTree::preorder(avl_node *node){
    if (node != NULL){
        cout << node->word << ": " << node->meaning << endl;
        preorder(node->left);
        preorder(node->right);}}
void avlTree::postorder(avl_node *node){
    if (node != NULL){
        postorder(node->left);
        postorder(node->right);
        cout << node->word << ": " << node->meaning << endl; }}
int main(){
    avlTree avl;
    int choice;
    string word, meaning;
do{
        cout << "\n1. Insert Word\n2. Display \n3. Search Word\n4. Update Meaning\n5. 
Exit\nEnter your choice: ";
        cin >> choice;
        cin.ignore();

        switch (choice)
        {
        case 1:
            avl.insert();
            break;
        case 2:
            avl.display();
            break;
        case 3:
            cout << "Enter word to search: ";
            cin >> word;
            avl.search(word);
            break;
        case 4:
            cout << "Enter word to update: ";
            cin >> word;
            cin.ignore();
            cout << "Enter new meaning: ";
            getline(cin, meaning);
            avl.update(word, meaning);
            break;
        case 5:
            cout << "Exiting the dictionary application. Bye!\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 5);

    return 0;}
```