#include <iostream>

#include <string>

using namespace std;

struct Node {

string keyword;

string meaning;

Node\* left;

Node\* right;

Node(string k, string m) : keyword(k), meaning(m), left(nullptr), right(nullptr) {}

};

class Dictionary {

private:

Node\* root;

Node\* insert(Node\* node, string keyword, string meaning) {

if (node == nullptr)

return new Node(keyword, meaning);

if (keyword < node->keyword)

node->left = insert(node->left, keyword, meaning);

else if (keyword > node->keyword)

node->right = insert(node->right, keyword, meaning);

return node;

}

void inorder(Node\* node) {

if (node != nullptr) {

inorder(node->left);

cout << node->keyword << ": " << node->meaning << endl;

inorder(node->right);

}

}

void reverse\_inorder(Node\* node) {

if (node != nullptr) {

reverse\_inorder(node->right);

cout << node->keyword << ": " << node->meaning << endl;

reverse\_inorder(node->left);

}

}

Node\* findMin(Node\* node) {

while (node->left != nullptr)

node = node->left;

return node;

}

Node\* deleteNode(Node\* node, string keyword) {

if (node == nullptr)

return node;

if (keyword < node->keyword)

node->left = deleteNode(node->left, keyword);

else if (keyword > node->keyword)

node->right = deleteNode(node->right, keyword);

else {

if (node->left == nullptr) {

Node\* temp = node->right;

delete node;

return temp;

} else if (node->right == nullptr) {

Node\* temp = node->left;

delete node;

return temp;

} else {

Node\* temp = findMin(node->right);

node->keyword = temp->keyword;

node->meaning = temp->meaning;

node->right = deleteNode(node->right, temp->keyword);

}

}

return node;

}

Node\* updateMeaning(Node\* node, string keyword, string newMeaning) {

if (node == nullptr)

return node;

if (keyword == node->keyword) {

node->meaning = newMeaning;

} else if (keyword < node->keyword) {

node->left = updateMeaning(node->left, keyword, newMeaning);

} else {

node->right = updateMeaning(node->right, keyword, newMeaning);

}

return node;

}

int maxComparisons(Node\* node, int depth) {

if (node == nullptr)

return depth;

int left = maxComparisons(node->left, depth + 1);

int right = maxComparisons(node->right, depth + 1);

return max(left, right);

}

public:

Dictionary() : root(nullptr) {}

void insert(string keyword, string meaning) {

root = insert(root, keyword, meaning);

}

void updateMeaning(string keyword, string newMeaning) {

root = updateMeaning(root, keyword, newMeaning);

cout << "Meaning of \"" << keyword << "\" updated!" << endl;

}

void deleteKeyword(string keyword) {

root = deleteNode(root, keyword);

cout << "Keyword \"" << keyword << "\" deleted!" << endl;

}

void displayAscending() {

cout << "Dictionary in Ascending Order:\n";

inorder(root);

}

void displayDescending() {

cout << "Dictionary in Descending Order:\n";

reverse\_inorder(root);

}

void findMaxComparisons() {

int comparisons = maxComparisons(root, 0);

cout << "Maximum comparisons required for searching a keyword: " << comparisons << endl;

}

};

int main() {

Dictionary dict;

int choice;

string keyword, meaning;

do {

cout << "\nDictionary Menu:\n";

cout << "1. Add a new keyword\n";

cout << "2. Update meaning of a keyword\n";

cout << "3. Delete a keyword\n";

cout << "4. Display dictionary in ascending order\n";

cout << "5. Display dictionary in descending order\n";

cout << "6. Find maximum comparisons for searching\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter keyword: ";

cin >> keyword;

cout << "Enter meaning: ";

cin.ignore();

getline(cin, meaning);

dict.insert(keyword, meaning);

break;

case 2:

cout << "Enter keyword to update: ";

cin >> keyword;

cout << "Enter new meaning: ";

cin.ignore();

getline(cin, meaning);

dict.updateMeaning(keyword, meaning);

break;

case 3:

cout << "Enter keyword to delete: ";

cin >> keyword;

dict.deleteKeyword(keyword);

break;

case 4:

dict.displayAscending();

break;

case 5:

dict.displayDescending();

break;

case 6:

dict.findMaxComparisons();

break;

case 7:

cout << "Exiting the program...\n";

break;

default:

cout << "Invalid choice! Please try again.\n";

}

} while (choice != 7);

return 0;

}