/\*A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation. \*/

#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 = right = nullptr;

}

};

class DictionaryBST {

private:

Node\* root;

// Helper function to insert a node

Node\* insert(Node\* root, string key, string meaning) {

if (!root) {

return new Node(key, meaning);

}

if (key < root->keyword)

root->left = insert(root->left, key, meaning);

else if (key > root->keyword)

root->right = insert(root->right, key, meaning);

else

cout << "Keyword already exists.\n";

return root;

}

// Helper function to display in order

void inOrder(Node\* root) {

if (!root) return;

inOrder(root->left);

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

inOrder(root->right);

}

// Helper function to display in reverse order

void reverseOrder(Node\* root) {

if (!root) return;

reverseOrder(root->right);

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

reverseOrder(root->left);

}

// Helper to find the minimum value node (used in deletion)

Node\* findMin(Node\* root) {

while (root && root->left)

root = root->left;

return root;

}

// Helper function to delete a node

Node\* deleteNode(Node\* root, string key) {

if (!root) return nullptr;

if (key < root->keyword)

root->left = deleteNode(root->left, key);

else if (key > root->keyword)

root->right = deleteNode(root->right, key);

else {

// Node with one or no child

if (!root->left) {

Node\* temp = root->right;

delete root;

return temp;

}

if (!root->right) {

Node\* temp = root->left;

delete root;

return temp;

}

// Node with two children

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

root->keyword = temp->keyword;

root->meaning = temp->meaning;

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

}

return root;

}

// Helper function to update meaning

void updateMeaning(Node\* root, string key, string newMeaning) {

if (!root) {

cout << "Keyword not found.\n";

return;

}

if (key < root->keyword)

updateMeaning(root->left, key, newMeaning);

else if (key > root->keyword)

updateMeaning(root->right, key, newMeaning);

else

root->meaning = newMeaning;

}

// Search and count comparisons

int search(Node\* root, string key, int& comparisons) {

if (!root) return 0;

comparisons++;

if (key == root->keyword)

return 1;

else if (key < root->keyword)

return search(root->left, key, comparisons);

else

return search(root->right, key, comparisons);

}

// Helper to calculate height (for max comparisons)

int height(Node\* root) {

if (!root) return 0;

return 1 + max(height(root->left), height(root->right));

}

public:

DictionaryBST() {

root = nullptr;

}

void addKeyword(string key, string meaning) {

root = insert(root, key, meaning);

}

void deleteKeyword(string key) {

root = deleteNode(root, key);

}

void updateKeyword(string key, string newMeaning) {

updateMeaning(root, key, newMeaning);

}

void displayAscending() {

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

inOrder(root);

}

void displayDescending() {

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

reverseOrder(root);

}

void findKeyword(string key) {

int comparisons = 0;

if (search(root, key, comparisons))

cout << "Keyword found with " << comparisons << " comparisons.\n";

else

cout << "Keyword not found after " << comparisons << " comparisons.\n";

}

void maxComparisons() {

cout << "Maximum comparisons in worst case: " << height(root) << endl;

}

};

int main() {

DictionaryBST dict;

int choice;

string keyword, meaning;

do {

cout << "\n------ Dictionary Menu ------\n";

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

cout << "2. Delete a keyword\n";

cout << "3. Update meaning of a keyword\n";

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

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

cout << "6. Search for a keyword\n";

cout << "7. Maximum comparisons in worst case\n";

cout << "8. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

cin.ignore(); // To handle the newline after choice input

switch (choice) {

case 1:

cout << "Enter keyword: ";

getline(cin, keyword);

cout << "Enter meaning: ";

getline(cin, meaning);

dict.addKeyword(keyword, meaning);

break;

case 2:

cout << "Enter keyword to delete: ";

getline(cin, keyword);

dict.deleteKeyword(keyword);

break;

case 3:

cout << "Enter keyword to update: ";

getline(cin, keyword);

cout << "Enter new meaning: ";

getline(cin, meaning);

dict.updateKeyword(keyword, meaning);

break;

case 4:

dict.displayAscending();

break;

case 5:

dict.displayDescending();

break;

case 6:

cout << "Enter keyword to search: ";

getline(cin, keyword);

dict.findKeyword(keyword);

break;

case 7:

dict.maxComparisons();

break;

case 8:

cout << "Exiting program.\n";

break;

default:

cout << "Invalid choice. Try again.\n";

}

} while (choice != 8);

return 0;

}