**Code**

#include <iostream>

using namespace std;

class node {

public:

string word;

string meaning;

node\* left = NULL;

node\* right = NULL;

node (string x, string y) {

word = x;

meaning = y;

left = NULL;

right = NULL;

}

friend class Dictionary;

};

class Dictionary {

public:

node\* root, \*q; //q is parent here

Dictionary () {

root = NULL;

q = NULL;

}

void insert (node\*, string, string);

void display\_asc (node \*);

void display\_desc (node \*);

void comparisons (node\*, string);

void updateWord (node\*, string);

void deleteWord (node\*, string);

node\* min\_node (node \*);

};

void Dictionary::insert (node\* p, string key, string keyMeaning) {

if (key < p -> word) {

if (p -> left != NULL)

insert (p -> left, key, keyMeaning);

else

p -> left = new node (key, keyMeaning);

}

else if (key > p -> word) {

if (p -> right != NULL)

insert (p -> right, key, keyMeaning);

else

p -> right = new node (key, keyMeaning);

}

}

void Dictionary::display\_asc (node \*p){ //inorder

if (p -> left != NULL)

display\_asc (p -> left);

cout << "\n" << p -> word << " \t" << p -> meaning;

if (p -> right != NULL)

display\_asc (p -> right);

}

void Dictionary::display\_desc (node \*p) {

if (p -> right != NULL)

display\_desc (p -> right);

cout << "\n" << p -> word << " \t" << p -> meaning;

if (p -> left != NULL)

display\_desc (p -> left);

}

void Dictionary::comparisons (node\* p, string key) {

static int count = 0;

while (p != NULL) {

if (key < p -> word) {

count++;

p = p -> left;

}

else if (key > p -> word) {

count++;

p = p -> right;

}

else if (key == p -> word) {

count++;

cout << "Number of comparisons to find the word: " << count;

return ;

}

}

cout << "\nWord not found!";

}

void Dictionary::deleteWord (node\* p, string key) {

node \*s;

while (p != NULL) { //searching for word

if (key < p -> word) {

q = p;

p = p -> left;

}

else if (key > p -> word) {

q = p;

p = p -> right;

}

else if (key == p -> word) { //word found

if (p -> left == NULL && p -> right == NULL) { //no child

if (q -> left == p) {

delete p;

q -> left = NULL;

return;

}

if (q -> right == p) {

delete p;

q -> right = NULL;

return;

}

}

if (p -> right != NULL && p -> left == NULL) { //right child only

if (q -> right == p) {

q -> right = p -> right;

delete p;

return;

}

else if (q -> left == p) {

q -> left = p -> right;

delete p;

return;

}

}

else if (p -> left != NULL && p -> right == NULL) { //left child only

if (q -> right == p) {

q -> right = p -> left;

delete p;

return;

}

else if (q -> left == p) {

q -> left = p -> left;

delete p;

return;

}

}

else if (p -> left != NULL && p -> right != NULL) {

s = min\_node (p -> right);

p -> word = s -> word;

p -> meaning = s -> meaning;

deleteWord (s, s -> word);

return;

}

}

}

cout << "\nWord NOT found!";

}

void Dictionary::updateWord (node\* p, string key) {

while (p != NULL) {

if (key < p -> word)

p = p -> left;

else if (key > p -> word)

p = p -> right;

else if (key == p -> word) {

cout << "\nEnter its new meaning: ";

cin >> p -> meaning;

return;

}

}

cout << "\nWord not found!";

}

node\* Dictionary::min\_node (node \*p) {

while (p -> left != NULL) {

q = p;

p = p -> left;

}

return p;

}

int main () {

int choice, n;

string newWord, searchWord, newMeaning;

Dictionary d1;

do {

cout << "\n\nDICTIONARY: "

<< "\n\n1. Insert new words"

<< "\n2. Display the dictionary in ascending order"

<< "\n3. Display the dictionary in descending order"

<< "\n4. Search and update a word"

<< "\n5. Delete a word"

<< "\n6. Comparisons"

<< "\n\nEnter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "\nEnter the number of words to insert: ";

cin >> n;

for (int i = 0 ; i < n ; i++) {

cout << "\nEnter the word to be inserted: ";

cin >> newWord;

cout << "\nEnter its meaning: ";

cin >> newMeaning;

if (d1.root == NULL)

d1. root = new node (newWord, newMeaning);

else

d1.insert (d1.root, newWord, newMeaning);

}

break;

case 2:

d1.display\_asc (d1.root);

break;

case 3:

d1.display\_desc (d1.root);

break;

case 4:

cout << "\nEnter the word to search: ";

cin >> searchWord;

d1.updateWord (d1.root, searchWord);

break;

case 5:

cout << "\nEnter the word to delete: ";

cin >> searchWord;

d1.deleteWord (d1.root, searchWord);

break;

case 6:

cout << "\nEnter the word to find comparisons: ";

cin >> searchWord;

d1.comparisons (d1.root, searchWord);

}

} while (choice < 7);

return 0;

}