**NAME** : Tejesh Santosh Yewale

**ROLL NO. :** A-61

**PRACTICAL NO. D2**

**CODE:**

#include <iostream>

using namespace std;

struct Node {

string keyword;

string meaning;

Node\* left;

Node\* right;

int height;

Node(string k, string m) {

keyword = k;

meaning = m;

left = right = NULL;

height = 1;

}

};

int height(Node\* root) {

return root ? root->height : 0;

}

int getBalance(Node\* root) {

if (!root) return 0;

return height(root->left) - height(root->right);

}

Node\* rightRotate(Node\* y) {

Node\* x = y->left;

Node\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node\* leftRotate(Node\* x) {

Node\* y = x->right;

Node\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

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 {

root->meaning = meaning; // update meaning

return root;

}

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

int balance = getBalance(root);

if (balance > 1 && key < root->left->keyword)

return rightRotate(root);

if (balance < -1 && key > root->right->keyword)

return leftRotate(root);

if (balance > 1 && key > root->left->keyword) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && key < root->right->keyword) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current->left != NULL)

current = current->left;

return current;

}

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

if (!root) return root;

if (key < root->keyword)

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

else if (key > root->keyword)

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

else {

if (!root->left || !root->right) {

Node\* temp = root->left ? root->left : root->right;

if (!temp) {

temp = root;

root = NULL;

} else

\*root = \*temp;

delete temp;

} else {

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

root->keyword = temp->keyword;

root->meaning = temp->meaning;

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

}

}

if (!root) return root;

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

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

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

comparisons++;

if (!root) return false;

if (key == root->keyword) return true;

else if (key < root->keyword) return search(root->left, key, comparisons);

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

}

void inorder(Node\* root) {

if (!root) return;

inorder(root->left);

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

inorder(root->right);

}

void reverseInorder(Node\* root) {

if (!root) return;

reverseInorder(root->right);

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

reverseInorder(root->left);

}

int maxComparisons(Node\* root) {

return height(root);

}

int main() {

Node\* root = NULL;

int choice;

string key, meaning;

do {

cout << "\n---- DICTIONARY MENU ----\n";

cout << "1. Insert keyword\n2. Delete keyword\n3. Search keyword\n4. Display Ascending\n5. Display Descending\n6. Max Comparisons(TREE HIGHT)\n0. Exit\nEnter your choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

cout << "Enter keyword: ";

getline(cin, key);

cout << "Enter meaning: ";

getline(cin, meaning);

root = insert(root, key, meaning);

break;

case 2:

cout << "Enter keyword to delete: ";

getline(cin, key);

root = deleteNode(root, key);

break;

case 3: {

cout << "Enter keyword to search: ";

getline(cin, key);

int comparisons = 0;

if (search(root, key, comparisons))

cout << "Found in " << comparisons << " comparisons.\n";

else

cout << "Not found after " << comparisons << " comparisons.\n";

break;

}

case 4:

cout << "\nDictionary (Ascending):\n";

inorder(root);

break;

case 5:

cout << "\nDictionary (Descending):\n";

reverseInorder(root);

break;

case 6:

cout << "\nMax Comparisons (Tree Height): " << maxComparisons(root) << endl;

break;

case 0:

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

break;

default:

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

}

} while (choice != 0);

return 0;

}

**OUTPUT:**





