**Lab Assignment No.9**

**Problem statement** : A Dictionary stores keywords and 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 Height balance tree and find the complexity for finding a keyword

**Code :**

#include <iostream>

#include <string>

using namespace std;

class dict {

dict\* left;

dict\* right;

string s1, s2;

public:

dict() : left(nullptr), right(nullptr) {}

void input();

void create\_node(dict\* node);

void check\_same(dict\* node);

void input\_display();

void display(dict\* tree, bool order);

void input\_remove();

dict\* remove(dict\* tree, const string& s3);

dict\* findmin(dict\* tree);

void input\_find();

dict\* find(dict\* tree, const string& s3);

void input\_update();

dict\* update(dict\* tree, const string& s3);

};

void dict::input() {

dict\* node = new dict;

cout << "\nEnter the keyword: ";

cin >> node->s1;

cout << "Enter the meaning of the keyword: ";

cin.ignore();

getline(cin, node->s2);

create\_node(node);

}

void dict::create\_node(dict\* node1) {

if (!left && !right) {

left = node1; // root node

cout << "\nRoot node created successfully.\n";

return;

}

check\_same(node1);

dict\* current = left;

dict\* parent = nullptr;

while (current) {

parent = current;

if (node1->s1 < current->s1)

current = current->left;

else if (node1->s1 > current->s1)

current = current->right;

else {

cout << "The word already exists.\n";

delete node1; // Avoid memory leak

return;

}

}

if (node1->s1 < parent->s1)

parent->left = node1;

else

parent->right = node1;

cout << "Node added.\n";

}

void dict::check\_same(dict\* node1) {

if (node1->s1 == left->s1) {

cout << "The word already exists.\n";

delete node1; // Avoid memory leak

}

}

void dict::input\_display() {

if (!left) {

cout << "No words in the dictionary.\n";

return;

}

bool order;

cout << "Enter 1 for ascending order, 0 for descending order: ";

cin >> order;

display(left, order);

}

void dict::display(dict\* tree, bool order) {

if (!tree) return;

if (order) {

display(tree->left, order);

cout << tree->s1 << " = " << tree->s2 << endl;

display(tree->right, order);

} else {

display(tree->right, order);

cout << tree->s1 << " = " << tree->s2 << endl;

display(tree->left, order);

}

}

void dict::input\_remove() {

if (!left) {

cout << "No words to delete.\n";

return;

}

cout << "Enter keyword to delete: ";

cin >> s1;

left = remove(left, s1);

cout << "Word deleted.\n";

}

dict\* dict::remove(dict\* tree, const string& s3) {

if (!tree) return nullptr;

if (s3 < tree->s1)

tree->left = remove(tree->left, s3);

else if (s3 > tree->s1)

tree->right = remove(tree->right, s3);

else {

if (!tree->left) {

dict\* temp = tree->right;

delete tree;

return temp;

} else if (!tree->right) {

dict\* temp = tree->left;

delete tree;

return temp;

}

dict\* temp = findmin(tree->right);

tree->s1 = temp->s1;

tree->s2 = temp->s2;

tree->right = remove(tree->right, temp->s1);

}

return tree;

}

dict\* dict::findmin(dict\* tree) {

while (tree->left)

tree = tree->left;

return tree;

}

void dict::input\_find() {

if (!left) {

cout << "No words to search.\n";

return;

}

cout << "Enter keyword to search: ";

cin >> s1;

find(left, s1);

}

dict\* dict::find(dict\* tree, const string& s3) {

if (!tree) {

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

return nullptr;

}

if (tree->s1 == s3) {

cout << "Found: " << tree->s1 << " = " << tree->s2 << endl;

return tree;

}

return s3 < tree->s1 ? find(tree->left, s3) : find(tree->right, s3);

}

void dict::input\_update() {

if (!left) {

cout << "No words to update.\n";

return;

}

cout << "Enter keyword to update: ";

cin >> s1;

update(left, s1);

}

dict\* dict::update(dict\* tree, const string& s3) {

dict\* found = find(tree, s3);

if (found) {

cout << "Enter new meaning: ";

cin.ignore();

getline(cin, found->s2);

cout << "Meaning updated.\n";

}

return tree;

}

int main() {

int ch;

dict d;

do {

cout << "\n1. Add new keyword\n2. Display dictionary\n3. Remove keyword\n4. Find keyword\n5. Update keyword\n6. Exit\n";

cin >> ch;

switch (ch) {

case 1: d.input(); break;

case 2: d.input\_display(); break;

case 3: d.input\_remove(); break;

case 4: d.input\_find(); break;

case 5: d.input\_update(); break;

case 6: cout << "Exiting...\n"; break;

default: cout << "Invalid choice, try again.\n";

}

} while (ch != 6);

return 0;

}