Rajalakshmi Engineering College

Name: Rakesh H
Email: 240701415@rajalakshmi.edu.in
Roll no: 240701415
Phone: 7305737702
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a messaging application, users maintain a contact list with names and corresponding phone numbers. Develop a program to manage this contact list using a dictionary implemented with hashing.

The program allows users to add contacts, delete contacts, and check if a specific contact exists. Additionally, it provides an option to print the contact list in the order of insertion.

Input Format

The first line consists of an integer n, representing the number of contact pairs to be inserted.

Each of the next n lines consists of two strings separated by a space: the name of the contact (key) and the corresponding phone number (value).

The last line contains a string k, representing the contact to be checked or removed.

Output Format

If the given contact exists in the dictionary:

1. The first line prints "The given key is removed!" after removing it.
2. The next n - 1 lines print the updated contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

If the given contact does not exist in the dictionary:

1. The first line prints "The given key is not found!".
2. The next n lines print the original contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 3
Alice 1234567890
Bob 9876543210
Charlie 4567890123
Bob

Output: The given key is removed!
Key: Alice; Value: 1234567890
Key: Charlie; Value: 4567890123

Answer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
   char key[11];
```

```
    char value[11];
    int is_active;
} Slot;

typedef struct {
    Slot* table;
    char** order;
    int table_size;
    int order_count;
} ContactList;

int hash_function(const char* key, int table_size) {
    int sum = 0;
    for (int i = 0; key[i] != '\0'; i++) {
        sum += key[i];
    }
    return sum % table_size;
}

ContactList* init_contact_list(int table_size) {
    ContactList* cl = (ContactList*)malloc(sizeof(ContactList));
    cl->table = (Slot*)malloc(table_size * sizeof(Slot));
    cl->order = (char**)malloc(50 * sizeof(char*));
    cl->table_size = table_size;
    cl->order_count = 0;

    for (int i = 0; i < table_size; i++) {
        cl->table[i].key[0] = '\0';
        cl->table[i].value[0] = '\0';
        cl->table[i].is_active = -1;
    }
    for (int i = 0; i < 50; i++) {
        cl->order[i] = NULL;
    }
    return cl;
}

// Insert a contact into the hash table and order list
void insert(ContactList* cl, const char* key, const char* value) {
    int index = hash_function(key, cl->table_size);
    int original_index = index;
    int count = 0;
    // Linear probing to find an empty or deleted slot
    while (cl->table[index].is_active != -1 && count < cl->table_size) {
        if (cl->table[index].is_active == 1 && strcmp(cl->table[index].key, key) == 0) {
            return; // Key already exists, no duplicate insertion
        }
        index = (index + 1) % cl->table_size;
        count++;
        if (index == original_index) return; // Table full
    }

    // Insert into hash table
    strcpy(cl->table[index].key, key);
    strcpy(cl->table[index].value, value);
    cl->table[index].is_active = 1;

    // Add to insertion order list
    cl->order[cl->order_count] = (char*)malloc(11 * sizeof(char));
    strcpy(cl->order[cl->order_count], key);
    cl->order_count++;
}

// Search for a contact
int search(ContactList* cl, const char* key) {
    int index = hash_function(key, cl->table_size);
    int original_index = index;
    int count = 0;

    while (cl->table[index].is_active != -1 && count < cl->table_size) {
        if (cl->table[index].is_active == 1 && strcmp(cl->table[index].key, key) == 0) {
            return index; // Found
        }
        index = (index + 1) % cl->table_size;
        count++;
        if (index == original_index) break;
    }
    return -1; // Not found
}

// Delete a contact
void delete_contact(ContactList* cl, const char* key) {
    int index = search(cl, key);
    if (index != -1) {
        cl->table[index].is_active = 0; // Mark as deleted
    }
}

// Print contact list in insertion order, skipping deleted contacts
void print_contacts(ContactList* cl) {
    for (int i = 0; i < cl->order_count; i++) {
        int index = search(cl, cl->order[i]);
        if (index != -1 && cl->table[index].is_active == 1) {
            printf("Key: %s; Value: %s\n", cl->table[index].key, cl->table[index].value);
        }
    }
}

// Free memory
void free_contact_list(ContactList* cl) {
    for (int i = 0; i < cl->order_count; i++) {
        free(cl->order[i]);
    }
    free(cl->order);
    free(cl->table);
    free(cl);
}

int main() {
    int n;
    scanf("%d", &n); // Read number of contacts

    // Initialize contact list with table size 2*n to reduce collisions
    ContactList* cl = init_contact_list(2 * n);

    // Read and insert contacts
    char name[11], phone[11];
    for (int i = 0; i < n; i++) {
        scanf("%s %s", name, phone);
        insert(cl, name, phone);
    }

    // Read key to search/remove
    char key[11];
    scanf("%s", key);
    // Check if key exists
    int found = search(cl, key);
    if (found != -1) {
        printf("The given key is removed!\n");
        delete_contact(cl, key);
        print_contacts(cl); // Print remaining contacts
    } else {
        printf("The given key is not found!\n");
        print_contacts(cl); // Print all contacts
    }

    // Clean up
    free_contact_list(cl);
    return 0;
}
```

Status : Correct ***Marks : 10/10***