Rajalakshmi Engineering College

Name: LAL SHIVAAN S L
Email: 240701285@rajalakshmi.edu.in
Roll no: 240701285
Phone: 8608375254
Branch: REC
Department: I CSE AH
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>

#define TABLE_SIZE 101
```

```
// Contact structure
typedef struct Contact {
    char name[20];
    char number[20];
    struct Contact* next; // For chaining in hash table
    struct Contact* order_next; // For maintaining insertion order
} Contact;

// Hash table and order list head
Contact* hash_table[TABLE_SIZE];
Contact* order_head = NULL;
Contact* order_tail = NULL;

// Hash function (simple sum of chars mod TABLE_SIZE)
int hash(char* key) {
    int hash = 0;
    for (int i = 0; key[i]; i++) {
        hash = (hash + key[i]) % TABLE_SIZE;
    }
    return hash;
}

// Insert contact
void insert(char* name, char* number) {
    int index = hash(name);

    // Create contact node
    Contact* new_contact = (Contact*)malloc(sizeof(Contact));
    strcpy(new_contact->name, name);
    strcpy(new_contact->number, number);
    new_contact->next = NULL;
    new_contact->order_next = NULL;

    // Insert into hash table (chaining)
    new_contact->next = hash_table[index];
    hash_table[index] = new_contact;

    // Insert into order list
    if (order_head == NULL) {
        order_head = order_tail = new_contact;
    } else {
        order_tail->order_next = new_contact;
```

```
        order_tail = new_contact;
    }
}

// Delete contact by name
int delete_contact(char* name) {
    int index = hash(name);
    Contact *curr = hash_table[index], *prev = NULL;

    // Search in hash table
    while (curr) {
        if (strcmp(curr->name, name) == 0) {
            // Remove from hash table
            if (prev) prev->next = curr->next;
            else hash_table[index] = curr->next;

            // Remove from order list
            Contact *o_curr = order_head, *o_prev = NULL;
            while (o_curr) {
                if (strcmp(o_curr->name, name) == 0) {
                    if (o_prev) o_prev->order_next = o_curr->order_next;
                    else order_head = o_curr->order_next;
                    if (order_tail == o_curr) order_tail = o_prev;
                    break;
                }
                o_prev = o_curr;
                o_curr = o_curr->order_next;
            }

            free(curr);
            return 1; // Deleted
        }
        prev = curr;
        curr = curr->next;
    }
    return 0; // Not found
}

// Print contacts
void print_contacts() {
    Contact* curr = order_head;
    while (curr) {
```

```
        printf("Key: %s; Value: %s\n", curr->name, curr->number);
        curr = curr->order_next;
    }
}

int main() {
    int n;
    char name[20], number[20], key[20];

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

    // Read contacts
    for (int i = 0; i < n; i++) {
        scanf("%s %s", name, number);
        insert(name, number);
    }

    // Read contact to check/remove
    scanf("%s", key);

    // Try to delete
    if (delete_contact(key)) {
        printf("The given key is removed!\n");
        print_contacts();
    } else {
        printf("The given key is not found!\n");
        print_contacts();
    }

    return 0;
}
```

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