Rajalakshmi Engineering College

Name: Sherwin G M
Email: 240701496@rajalakshmi.edu.in
Roll no: 240701496
Phone: 7708605966
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>
#define MAX_SIZE 100

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

```
    char value[12];
    int isDeleted;
} Contact;

Contact hashTable[MAX_SIZE];
char *insertionOrder[MAX_SIZE];
int size = 0;
int capacity = MAX_SIZE;
int orderSize = 0;

int custom_strcmp(const char *s1, const char *s2) {
    while (*s1 && (*s1 == *s2)) {
        s1++;
        s2++;
    }
    return *(unsigned char *)s1 - *(unsigned char *)s2;
}

void custom_strcpy(char *dest, const char *src) {
    while ((*dest++ = *src++));
}

char *custom_strdup(const char *s) {
    size_t len = 0;
    const char *p = s;
    while (*p++) len++;
    char *new_s = (char *)malloc(len + 1);
    if (new_s == NULL) return NULL;
    custom_strcpy(new_s, s);
    return new_s;
}

int hashFunction(const char *key) {
    int hash = 0;
    const char *p = key;
    while (*p) {
        hash += *p++;
    }
    return hash % capacity;
}

void insertContact(const char *key, const char *value) {
    int index = hashFunction(key);
    int originalIndex = index;
    while (hashTable[index].key[0] != '\0' && !hashTable[index].isDeleted) {
        if (custom_strcmp(hashTable[index].key, key) == 0) {
            custom_strcpy(hashTable[index].value, value);
            return;
        }
        index = (index + 1) % capacity;
        if (index == originalIndex) {
            return; // Table is full
        }
    }
    custom_strcpy(hashTable[index].key, key);
    custom_strcpy(hashTable[index].value, value);
    hashTable[index].isDeleted = 0;
    insertionOrder[orderSize] = custom_strdup(key);
    if (insertionOrder[orderSize] == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    orderSize++;
    size++;
}

int deleteContact(const char *key) {
    int index = hashFunction(key);
    int originalIndex = index;
    while (hashTable[index].key[0] != '\0') {
        if (!hashTable[index].isDeleted && custom_strcmp(hashTable[index].key,
key) == 0) {
            hashTable[index].isDeleted = 1;
            for (int i = 0; i < orderSize; i++) {
                if (custom_strcmp(insertionOrder[i], key) == 0) {
                    free(insertionOrder[i]);
                    for (int j = i; j < orderSize - 1; j++) {
                        insertionOrder[j] = insertionOrder[j + 1];
                    }
                    orderSize--;
                    break;
                }
            }
            size--;
            return 1;
        }
        index = (index + 1) % capacity;
        if (index == originalIndex) {
            break;
        }
    }
    return 0;
}

int searchContact(const char *key) {
    int index = hashFunction(key);
    int originalIndex = index;
    while (hashTable[index].key[0] != '\0') {
        if (!hashTable[index].isDeleted && custom_strcmp(hashTable[index].key,
key) == 0) {
            return 1;
        }
        index = (index + 1) % capacity;
        if (index == originalIndex) {
            break;
        }
    }
    return 0;
}

void printContacts() {
    for (int i = 0; i < orderSize; i++) {
        const char *key = insertionOrder[i];
        int index = hashFunction(key);
        int originalIndex = index;
        while (hashTable[index].key[0] != '\0') {
            if (!hashTable[index].isDeleted && custom_strcmp(hashTable[index].key,
key) == 0) {
                printf("Key: %s; Value: %s\n", hashTable[index].key,
hashTable[index].value);
                break;
            }
            index = (index + 1) % capacity;
            if (index == originalIndex) {
                break;
            }
        }
    }
}

int main() {
    int n;
    if (scanf("%d", &n) != 1) {
        fprintf(stderr, "Failed to read the number of contacts\n");
        return 1;
    }
    char key[11], value[12];
    for (int i = 0; i < n; i++) {
        if (scanf("%10s %11s", key, value) != 2) {
            fprintf(stderr, "Failed to read contact %d\n", i + 1);
            return 1;
        }
        insertContact(key, value);
    }
    char k[11];
    if (scanf("%10s", k) != 1) {
        fprintf(stderr, "Failed to read the key to search/delete\n");
        return 1;
    }
    if (searchContact(k)) {
        deleteContact(k);
        printf("The given key is removed!\n");
    } else {
        printf("The given key is not found!\n");
    }
    printContacts();
    for (int i = 0; i < orderSize; i++) {
        free(insertionOrder[i]);
    }
    return 0;
}
```

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