#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define LIST\_LEN 3

#define STRING\_SIZE 50

struct record\* new\_record(char\* last\_name, char\* first\_name);

struct record\* append\_record(int index, char\* last\_name, char\* first\_name);

unsigned int find\_hash(char\* s);

unsigned int find\_double\_hash(char\* s, int collisions, unsigned int hash);

void add\_record(char\* last\_name, char\* first\_name);

void display\_record(char\* last\_name, char\* first\_name);

struct record{

    int num;

    char first\_name[STRING\_SIZE];

    char last\_name[STRING\_SIZE];

    struct record\* next;

};

int num\_terms = 0;

struct record\* records[LIST\_LEN];

int main(){

    while(1){

        printf("\nPhone Directory\n");

        printf("[1] Add Record\n");

        printf("[2] Search Record\n");

        printf("[3] Exit\n");

        printf("Select one of the above options: ");

        int option;

        scanf("%d", &option);

        char first\_name[STRING\_SIZE];

        char last\_name[STRING\_SIZE];

        printf("\n");

        switch(option) {

            case 1:

                printf("Enter the Person's First Name: ");

                scanf("%s", first\_name);

                printf("Enter the Person's Last Name: ");

                scanf("%s", last\_name);

                add\_record(last\_name, first\_name);

                break;

            case 2:

                printf("Enter Person's First Name to search for: ");

                scanf("%s", first\_name);

                printf("Enter Person's Last Name to search for: ");

                scanf("%s", last\_name);

                display\_record(last\_name, first\_name);

                break;

            case 3:

                exit(0);

                break;

            default:

                printf("Enter a valid option.");

        }

        printf("\n");

    }

    return 0;

}

struct record\* new\_record(char\* last\_name, char\* first\_name){

    struct record\* rec = (struct record\*)malloc(sizeof(struct record));

    strncpy(rec->last\_name, last\_name, STRING\_SIZE);

    strncpy(rec->first\_name, first\_name, STRING\_SIZE);

    printf("Enter the Person's Number: ");

    scanf("%d", &rec->num);

    return rec;

}

struct record\* append\_record(int index, char\* last\_name, char\* first\_name){

    struct record\* rec = (struct record\*)malloc(sizeof(struct record));

    struct record\* start = records[index];

    if(strcmp(start->last\_name, last\_name) == 0 && strcmp(start->first\_name, first\_name) == 0){

        printf("Enter the Person's Number: ");

        int num;

        scanf("%d", &num);

        if(num == start->num){

            printf("\nThis record already exists.");

            free(rec);

            return start;

        }

        strncpy(rec->last\_name, last\_name, STRING\_SIZE);

        strncpy(rec->first\_name, first\_name, STRING\_SIZE);

        rec->num = num;

        rec->next = start;

        records[index] = rec;

        printf("\nRecord added Successfully.");

        return records[index];

    }

    rec = new\_record(last\_name, first\_name);

    struct record\* temp = start;

    while(temp->next != NULL){

        temp = temp->next;

    }

    temp->next = rec;

    return start;

}

unsigned int find\_hash(char\* s) {

    unsigned int hash = 0;

    while (\*s) {

        hash = (hash \* LIST\_LEN) + \*s;

        s++;

    }

    return hash % LIST\_LEN;

}

unsigned int find\_double\_hash(char\* s, int collisions, unsigned int hash) {

    return (hash + collisions \* (1 + (hash % (LIST\_LEN - 1))) + collisions) % LIST\_LEN;

}

void add\_record(char\* last\_name, char\* first\_name){

    int index = find\_hash(last\_name);

    int collisions = 1;

    if(num\_terms == LIST\_LEN){

        int original\_index = index;

        while(strcmp(records[index]->last\_name, last\_name) != 0){

            if(collisions == LIST\_LEN){

                records[index] = append\_record(original\_index, last\_name, first\_name);

                return;

            }

            index = find\_double\_hash(last\_name, collisions, index) % LIST\_LEN;

            ++collisions;

        }

        records[index] = append\_record(index, last\_name, first\_name);

        return;

    }

    while(records[index] != NULL){

        if(strcmp(records[index]->last\_name, last\_name) == 0){

            records[index] = append\_record(index, last\_name, first\_name);

            return;

        }

        index = find\_double\_hash(last\_name, collisions, index) % LIST\_LEN;

        ++collisions;

    }

    struct record\* rec = new\_record(last\_name, first\_name);

    records[index] = rec;

    printf("\nRecord added Successfully.");

    num\_terms++;

}

int search\_list(struct record\* temp, char\* last\_name, char\* first\_name){

    int found = 0;

    while(temp){

        if(strcmp(temp->last\_name, last\_name) == 0 && strcmp(temp->first\_name, first\_name) == 0){

            printf("\n%d", temp->num);

            found = 1;

        }

        temp = temp->next;

    }

    return found;

}

void display\_record(char\* last\_name, char\* first\_name){

    int index = find\_hash(last\_name);

    int collisions = 1;

    int found = 0;

    if(LIST\_LEN == num\_terms){

        int original\_index = index;

        while(strcmp(records[index]->last\_name, last\_name) != 0){

            if(collisions == LIST\_LEN){

                found = search\_list(records[original\_index], last\_name, first\_name);

                if(!found)

                    printf("\nNo matching records found for this name.");

                return;

            }

            index = find\_double\_hash(last\_name, collisions, index) % LIST\_LEN;

            ++collisions;

        }

        int secondary\_found = search\_list(records[index], last\_name, first\_name);

        found = (found || secondary\_found);

    }

    while(records[index] != NULL){

        if(strcmp(records[index]->last\_name, last\_name) == 0){

            found = search\_list(records[index], last\_name, first\_name);

            return;

        }

        index = find\_double\_hash(last\_name, collisions, index) % LIST\_LEN;

        ++collisions;

    }

    printf("\nNo matching records found for this name.");

}