Hash table implementation with help of linear probing technique for avoiding collision:

Code implementation:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int tableSize = 0, totEle = 0;

struct node \*hashTable = NULL;

struct node {

long int Mobile\_number;

int key;

char name[100];

char email[100];

int marker;

};

void insertInHash(int key, char \*name,char \*email, long int Mobile\_number) {

int hashIndex = key % tableSize;

if (tableSize == totEle) {

printf("Can't perform Insertion..Hash Table is full!!");

return;

}

while (hashTable[hashIndex].marker == 1) {

hashIndex = (hashIndex + 1)%tableSize;

}

hashTable[hashIndex].key = key;

hashTable[hashIndex].Mobile\_number = Mobile\_number;

strcpy(hashTable[hashIndex].name, name);

strcpy(hashTable[hashIndex].email, email);

hashTable[hashIndex].marker = 1;

totEle++;

return;

}

void deleteFromHash(int key) {

int hashIndex = key % tableSize, count = 0, flag = 0;

if (totEle == 0) {

printf("Hash Table is Empty!!\n");

return;

}

while (hashTable[hashIndex].marker != 0 && count <= tableSize) {

if (hashTable[hashIndex].key == key) {

hashTable[hashIndex].key = 0;

hashTable[hashIndex].marker = -1;

hashTable[hashIndex].Mobile\_number = 0;

strcpy(hashTable[hashIndex].name, "\0");

strcpy(hashTable[hashIndex].email, "\0");

totEle--;

flag = 1;

break;

}

hashIndex = (hashIndex + 1)%tableSize;

count++;

}

if (flag)

printf("Given data deleted from Hash Table\n");

else

printf("Given data is not available in Hash Table\n");

return;

}

void searchElement(int key) {

int hashIndex = key % tableSize, flag = 0, count = 0;

if (totEle == 0) {

printf("Hash Table is Empty!!");

return;

}

while (hashTable[hashIndex].marker != 0 && count <= tableSize) {

if (hashTable[hashIndex].key == key) {

printf("Name : %s\n", hashTable[hashIndex].name);

printf("Email Id : %s\n", hashTable[hashIndex].email);

printf("Mobile Number : %ld\n", hashTable[hashIndex].Mobile\_number);

flag = 1;

break;

}

hashIndex = (hashIndex + 1)%tableSize;

}

if (!flag)

printf("Given data is not present in hash table\n");

return;

}

void display() {

int i;

if (totEle == 0) {

printf("Hash Table is Empty!!\n");

return;

}

printf("Name Mobile Number Email Index \n");

printf("-----------------------------------------\n");

for (i = 0; i < tableSize; i++) {

if (hashTable[i].marker == 1) {

printf("%-15s", hashTable[i].name);

printf("%-13ld", hashTable[i].Mobile\_number);

printf("%-20s", hashTable[i].email);

printf("%d\n", i);

}

}

printf("\n");

return;

}

int main() {

int key=0, ch;

long int Mobile\_number,x;

char name[100];

char email[100];

printf("Enter the no of elements:");

scanf("%d", &tableSize);

hashTable = (struct node \*)calloc(tableSize, sizeof(struct node));

while (1) {

printf("1. Insertion\t2. Deletion\n");

printf("3. Searching\t4. Display\n");

printf("5. Exit\nEnter ur choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter your Name:");

scanf("%s",name);

printf("Enter your email:");

scanf("%s",email);

printf("Enter your Mobile number:");

scanf("%ld", &Mobile\_number);

x=Mobile\_number;

key=0;

while(x!=0)

{

int r=x%10;

key+=r;

x/=10;

}

insertInHash(key, name,email, Mobile\_number);

break;

case 2:

printf("Enter the Mobile\_number:");

scanf("%ld", &Mobile\_number);

x=Mobile\_number;

key=0;

while(x!=0)

{

int r=x%10;

key+=r;

x/=10;

}

deleteFromHash(key);

break;

case 3:

printf("Enter the Mobile\_number:");

scanf("%ld", &Mobile\_number);

x=Mobile\_number;

key=0;

while(x!=0)

{

int r=x%10;

key+=r;

x/=10;

}

searchElement(key);

break;

case 4:

display();

break;

case 5:

exit(0);

default:

printf("U have entered wrong Option!!\n");

break;

}

}

return 0;

}