**Experiment No.12**

**Aim:** Implementation of hashing functions with different collision resolution techniques.

**Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct hash \*hashTable = NULL;

int eleCount = 0;

struct node {

int key, age;

char name[100];

struct node \*next;

};

struct hash {

struct node \*head;

int count;

};

struct node \* createNode(int key, char \*name, int age) {

struct node \*newnode;

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->key = key;

newnode->age = age;

strcpy(newnode->name, name);

newnode->next = NULL;

return newnode;

}

void insertToHash(int key, char \*name, int age)

{

int hashIndex = key % eleCount;

struct node \*newnode = createNode(key, name, age);

/\* head of list for the bucket with index "hashIndex" \*/

if (!hashTable[hashIndex].head) {

hashTable[hashIndex].head = newnode;

hashTable[hashIndex].count = 1;

return;

}

/\* adding new node to the list \*/

newnode->next = (hashTable[hashIndex].head);

/\*

\* update the head of the list and no of nodes in the current bucket

\*/

hashTable[hashIndex].head = newnode;

hashTable[hashIndex].count++;

return;

}

void deleteFromHash(int key) {

/\* find the bucket using hash index \*/

int hashIndex = key % eleCount, flag = 0;

struct node \*temp, \*myNode;

/\* get the list head from current bucket \*/

myNode = hashTable[hashIndex].head;

if (!myNode) {

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

return;

}

temp = myNode;

while (myNode != NULL) {

/\* delete the node with given key \*/

if (myNode->key == key) {

flag = 1;

if (myNode == hashTable[hashIndex].head)

hashTable[hashIndex].head = myNode->next;

else

temp->next = myNode->next;

hashTable[hashIndex].count--;

free(myNode);

break;

}

temp = myNode;

myNode = myNode->next;

}

if (flag)

printf("Data deleted successfully from Hash Table\n");

else

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

return;

}

void searchInHash(int key) {

int hashIndex = key % eleCount, flag = 0;

struct node \*myNode;

myNode = hashTable[hashIndex].head;

if (!myNode) {

printf("Search element unavailable in hash table\n");

return;

}

while (myNode != NULL) {

if (myNode->key == key) {

printf("VoterID : %d\n", myNode->key);

printf("Name : %s\n", myNode->name);

printf("Age : %d\n", myNode->age);

flag = 1;

break;

}

myNode = myNode->next;

}

if (!flag)

printf("Search element unavailable in hash table\n");

return;

}

void display() {

struct node \*myNode;

int i;

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

if (hashTable[i].count == 0)

continue;

myNode = hashTable[i].head;

if (!myNode)

continue;

printf("\nData at index %d in Hash Table:\n", i);

printf("VoterID Name Age \n");

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

while (myNode != NULL) {

printf("%-12d", myNode->key);

printf("%-15s", myNode->name);

printf("%d\n", myNode->age);

myNode = myNode->next;

}

}

return;

}

int main() {

int n, ch, key, age;

char name[100];

printf("Enter the number of elements:");

scanf("%d", &n);

eleCount = n;

/\* create hash table with "n" no of buckets \*/

hashTable = (struct hash \*)calloc(n, sizeof (struct hash));

while (1) {

printf("\n1. Insertion\t2. Deletion\n");

printf("3. Searching\t4. Display\n5. Exit\n");

printf("Enter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the key value:");

scanf("%d", &key);

getchar();

printf("Name:");

fgets(name, 100, stdin);

name[strlen(name) - 1] = '\0';

printf("Age:");

scanf("%d", &age);

/\*inserting new node to hash table \*/

insertToHash(key, name, age);

break;

case 2:

printf("Enter the key to perform deletion:");

scanf("%d", &key);

/\* delete node with "key" from hash table \*/

deleteFromHash(key);

break;

case 3:

printf("Enter the key to search:");

scanf("%d", &key);

searchInHash(key);

break;

case 4:

display();

break;

case 5:

exit(0);

default:

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

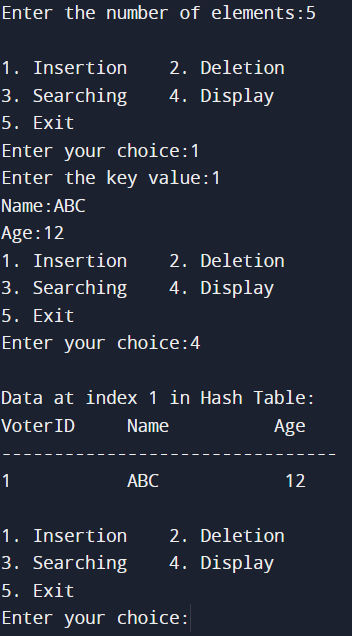
break;

}

}

return 0;

}



**Conclusion:**

In this way implementation of hash table along with hash function is studied. Also how to resolve collision using different techniques like chaining, linear probing, quadratic probing and double probing is understood.