**SHELL SCRIPT**

**EXPT NO: 4 DATE:**

**AIM**

Write a shell script program to implement a Hash Table

**THEORY**

Hashing is a procedure in which the key is converted into and integer value within a range, and this converted value can be used as an index of an array. The process of converting a key into an address is known as Hashing or key to address transformation.

A hashing function is used to generate an address from a key. The key may be of any type like integer or string but the resulting hash value will always be an integer.



We represent this as

**h(k)=a**

**h: -** Hash Function

**k: -** Key

**a: -** has value of k

Now the key can be stored at array index a, which is known as the home address. The key and the in which the insertion and searching is done through hashing is called hash table.

**Hash Function**

If the size of hash table is m, then we need a hash function that can generate addresses between 0 to n-1.

Two criteria to create a good has function

1) It should be easy to computer

2) It should generate addresses with minimum collision i.e., it should distribute the keys as uniformly as possible in the hash table.

**Collison Resolution**

An ideal hash function should perform one to one mapping between set of all possible keys and all hash table addresses but this is almost impossible and no hash function can totally prevent collisions. A collision occurs whenever a key is mapped to an address that is already occupied, and the different resolution suggest for an alternate place where this key can be placed.

The two collision resolution techniques are: -

**1)** Open Addressing (Closed Hashing)

**2)** Separate chaining (Open Hashing)

In the process of searching, the given key is compared with many keys and each key comparison is known as a probe. The efficiency of a collision technique is defined in terms of the number of probes required to find a record within a given key.

**Open Addressing (Closed Hashing)**

In open addressing, initially a key value is mapped to a particular address in the hash table. If that address id occupied then we will try to insert the key in some other empty location inside the table. The array is assumed to be closed and hence this method is named as closed hashing.

The three methods are: -

1) Linear Probing

2) Quadratic Probing

3) Double Hashing

**Linear Probing**

If the address given by the hash function is already occupied, then the key will be inserted in the next empty location in the hash table. If the address given by the hash function is ‘a’ and it is not empty, then we will try to insert the key in the next location. i.e., address a+1. If address a+1 is also occupied then then wee will try to insert in the next location (a+2) and we will keep on trying successive locations till we find an empty location where the can be inserted.

The Formula for Linear Probing is: -

**H(k,i)=(h(k)+i) mod Tsize (**Here i varies from 0 to Tsize-1)

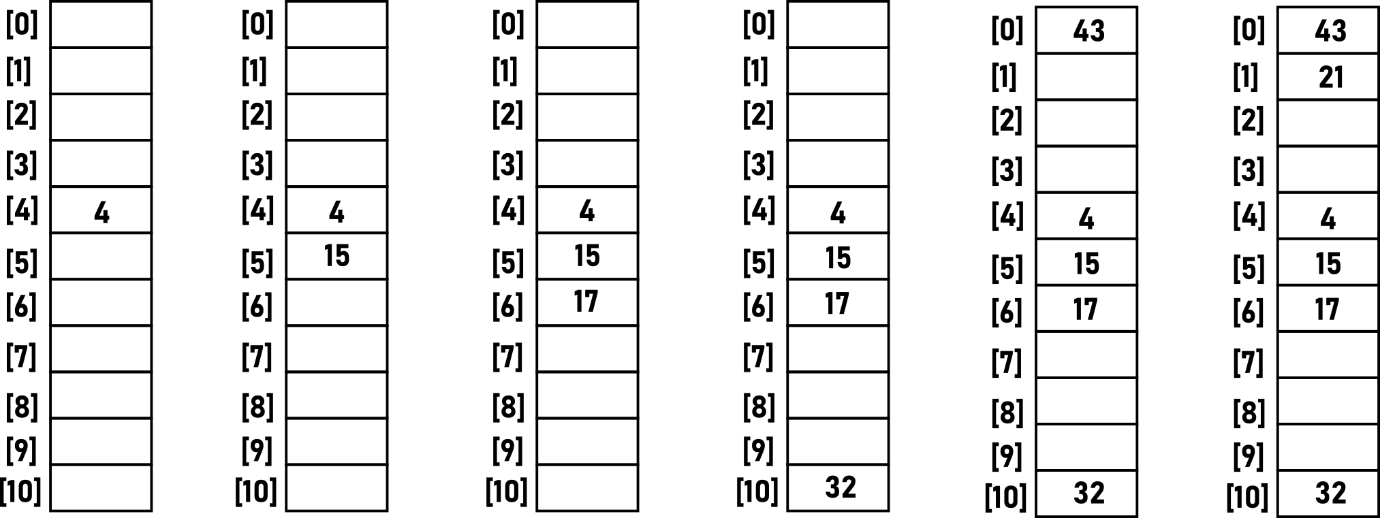
Consider the Following Example

Insert Keys: 4, 15, 17, 32, 43, 21

h(4)=4%11=4

h(15)=15%11=4 \*Collision 1

h(17)=17%11=6

h(32)=32%11=10

h(43)=43%11=10 \*Collision 2

h(21)=21%11=10 \*Collision 3

h(15)=15%11=4 \*Collision 1

H(15)=(h(15)+1)%11=5

h(43)=43%11=10 \*Collision 2

H(43)=(h(43)+1)%11=0

h(21)=21%11=10 \*Collision 3

H(21)=(h(21)+1)%11=0

H(21)=(h(21)+2)%11=1

**CODE**

**hashTab=(-1 -1 -1 -1 -1 -1 -1 -1 -1 -1)**

**value=0**

**function Insert(){**

**echo 'ENTER NUMBER TO BE INSERTED'**

**read NUM**

**HashValue=$(($NUM%10))**

**for (( i=0; i<10; i++ ))**

**do**

**index=$((($HashValue+$i)%10))**

**B=${hashTab[$index]}**

**if [ $B -eq -1 ]**

**then**

**hashTab[$index]=$NUM**

**break**

**fi**

**if [ $i -eq 9 ]**

**then**

**echo "ELEMENT CANNOT BE INSERTED"**

**fi**

**done**

**}**

**function Search(){**

**echo 'ENTER NUMBER TO BE SEARCHED'**

**read NUM**

**HashValue=$(($NUM%10))**

**for (( i=0; i<10; i++ ))**

**do**

**index=$((($HashValue+$i)%10))**

**B=${hashTab[$index]}**

**if [ $B -eq $NUM ]**

**then**

**echo "NUMBER PRESENT IN HASH TABLE"**

**break**

**fi**

**if [ $i -eq 9 ]**

**then**

**echo "ELEMENT NOT PRESENT"**

**fi**

**done**

**}**

**function Delete(){**

**echo 'ENTER NUMBER TO BE DELETED'**

**read NUM**

**HashValue=$(($NUM%10))**

**for (( i=0; i<10; i++ ))**

**do**

**index=$((($HashValue+$i)%10))**

**B=${hashTab[$index]}**

**if [ $B -eq $NUM ]**

**then**

**hashTab[$index]=-1**

**echo "NUMBER DELETED"**

**break**

**fi**

**if [ $i -eq 9 ]**

**then**

**echo "ELEMENT NOT PRESENT"**

**fi**

**done**

**}**

**function Display(){**

**echo "HASH CONTENT: "**

**for (( i=0; i<=10; i++ ))**

**do**

**a=${hashTab[$i]}**

**echo "$a"**

**done**

**}**

**ch=1**

**while [ $ch -eq 1 ]**

**do**

**echo "1. INSERTED"**

**echo "2. SEARCH"**

**echo "3. DISPLAY"**

**echo "4. DELETE"**

**read choice**

**case $choice in**

**1) Insert**

**;;**

**2) Search**

**;;**

**3) Display**

**;;**

**4) Delete**

**;;**

**\*) echo "INVALID CHOICE"**

**;;**

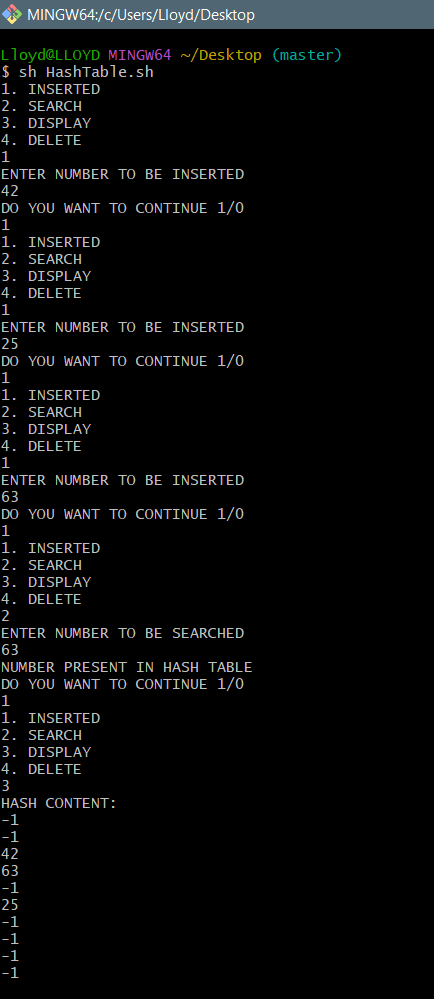
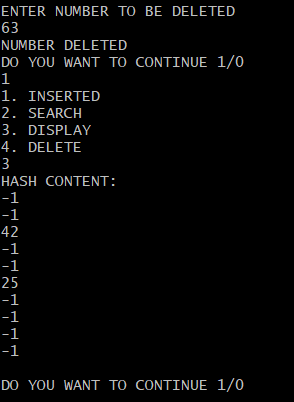
**esac**

**echo "DO YOU WANT TO CONTINUE 1/0"**

**read ch**

**done**

**OUTPUT**

****

**CONCLUSION**

The given problem statement was successfully implemented.