K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: D2	**Roll. No.: 16010122323**
Experiment:	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of hashing concept

Objective: To understand various hashing methods

Expected Outcome of Experiment:

CO	Outcome
CO4	Demonstrate sorting and searching methods.

Websites/books referred:

1. geeksforgeeks.com

2.Reema Thareja- Data structures.

Abstract: -

HASHING

Hashing is a technique used in computer science to convert data (usually of variable size) into a fixed-size string of characters or a numerical value. The result of this conversion is known as a hash code or hash value. Hashing is commonly used in data structures and algorithms to quickly locate a data record in a table or to secure data.

HASH FUNCTION

A hash function is a mathematical function or algorithm that takes an input (or "key") and returns a fixed-size string of characters or a numerical value, which is typically a hash code. The key can be of variable length, but the hash code is of a fixed size. A good hash function should have the following properties:

- Deterministic: Given the same input, it should always produce the same hash code.
- Fast to compute.
- The hash codes should be distributed uniformly across the possible hash values.
- Small changes in the input should result in significantly different hash codes (avalanche effect).

LIST COLLISION HANDLING METHODS

Collision handling methods are techniques used to deal with collisions that occur when two different keys produce the same hash code. Collisions are inevitable in hashing due to the finite range of possible hash codes and the potentially infinite number of keys. These collision handling methods are used in hash tables to ensure that data can be efficiently stored and retrieved, even when multiple keys map to the same hash code. The choice of method depends on the specific requirements and constraints of the application. Common collision handling methods include:

- **Linear Probing:** When a collision occurs, this method checks the next available slot in a linear fashion until an empty slot is found.
- **Quadratic Hashing:** Similar to linear probing but uses a quadratic sequence to search for the next available slot.
- **Chaining:** In this method, each bucket (or slot) in the hash table contains a linked list of elements. Collisions are resolved by adding elements to the linked list associated with the corresponding bucket.
- **Double Hashing:** This method uses a second hash function to determine the interval between probes when collisions happen, which can reduce clustering.

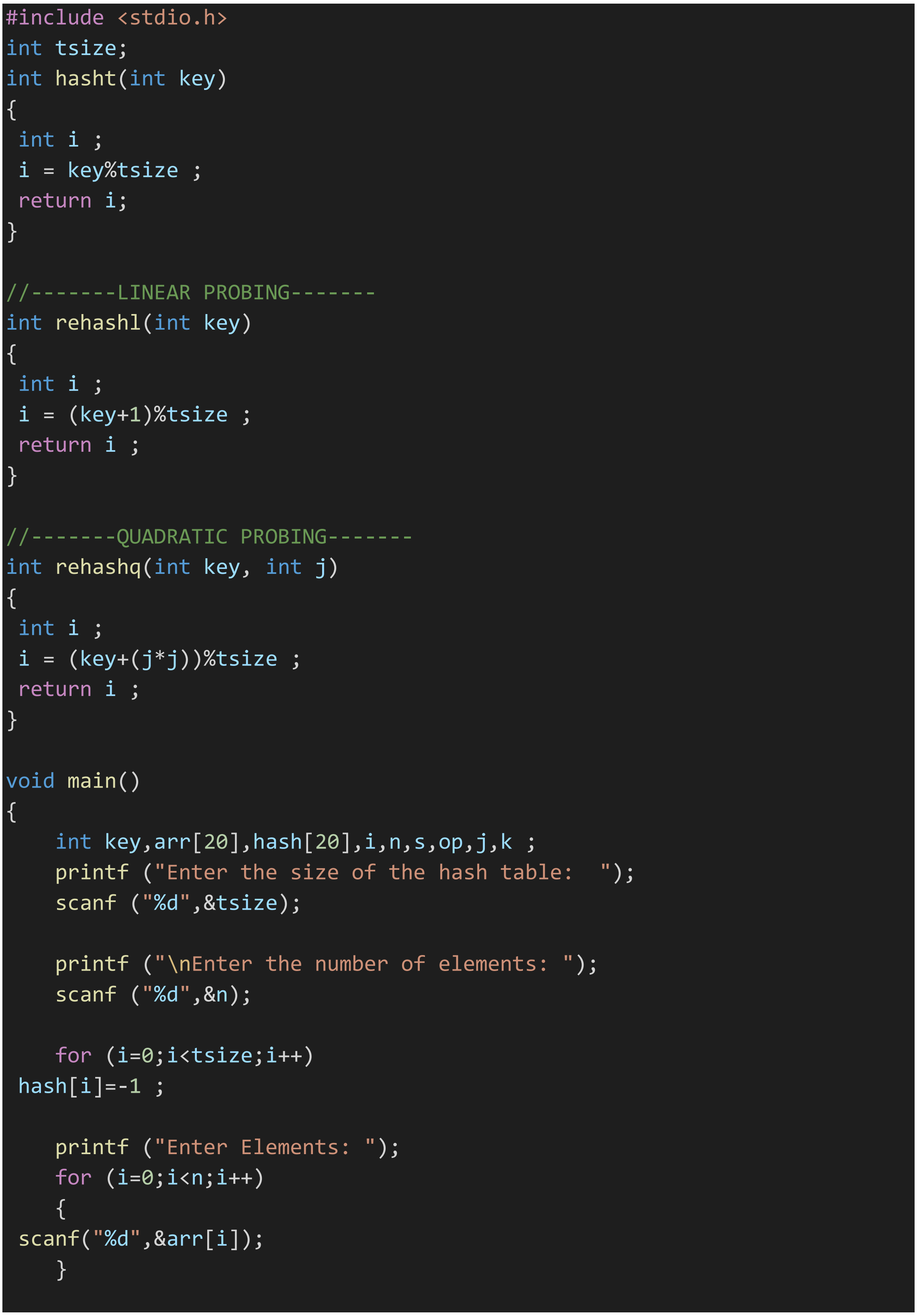
- **Open Addressing:** Open addressing is a general term for collision resolution methods that involve searching for the next available slot in the hash table. Linear probing, quadratic hashing, and double hashing are specific forms of open addressing.
- **Cuckoo Hashing:** This method involves two or more hash tables and relocates keys to different tables when collisions occur, making it suitable for scenarios with low load factors.

EXAMPLE:

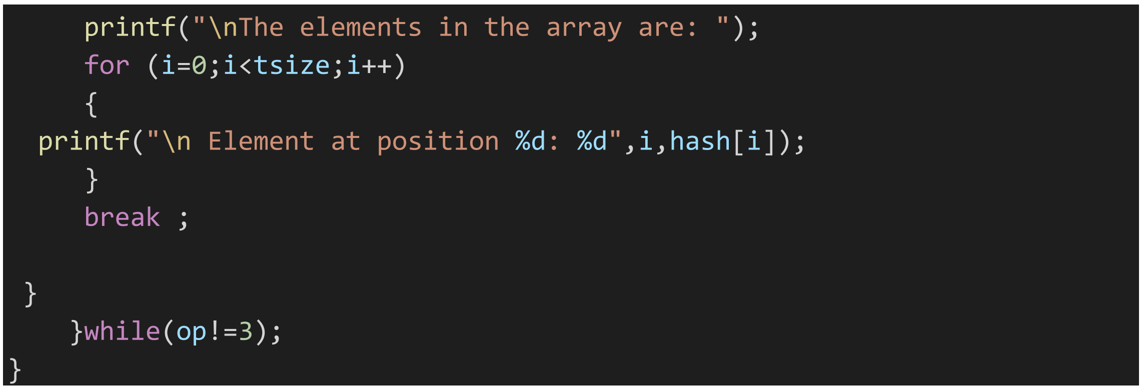
CD Databases For CDs, it is desirable to have a world-wide CD database so that when users put their disk in the CD player, they get a full table of contents on their own computer's screen. These tables are not stored on the disks themselves, i.e., the CD does not store any information about the songs, rather this information is downloaded from the database. The critical issue to solve here is that CDs have no ID numbers stored on them, so how will the computer know which CD has been put in the player? The only information that can be used is the track length, and the fact that every CD is different. Basically, a big number is created from the track lengths, also known as a 'signature'. This signature is used to identify a particular CD. The signature is a value obtained by hashing. For example, a number of length of 8 or 10 hexadecimal. digits is made up; the number is then sent to the database, and that database looks for the closest match. The reason being that track length may not be measured exactly.

File Signatures File signatures provide a compact means of identifying files. We use a function, h[x], the file signature, which is a property of the file. Although we can store files by name, signatures provide a compact identity to files.
Since a signature depends on the contents of a file, if any change is made to the file, then the signature will change. In this way, the signature of a file can be used as a quick verification to see if anyone has altered the file, or if it has lost a bit during transmission.
Signatures are widely used for files that store marks of students

CODE:

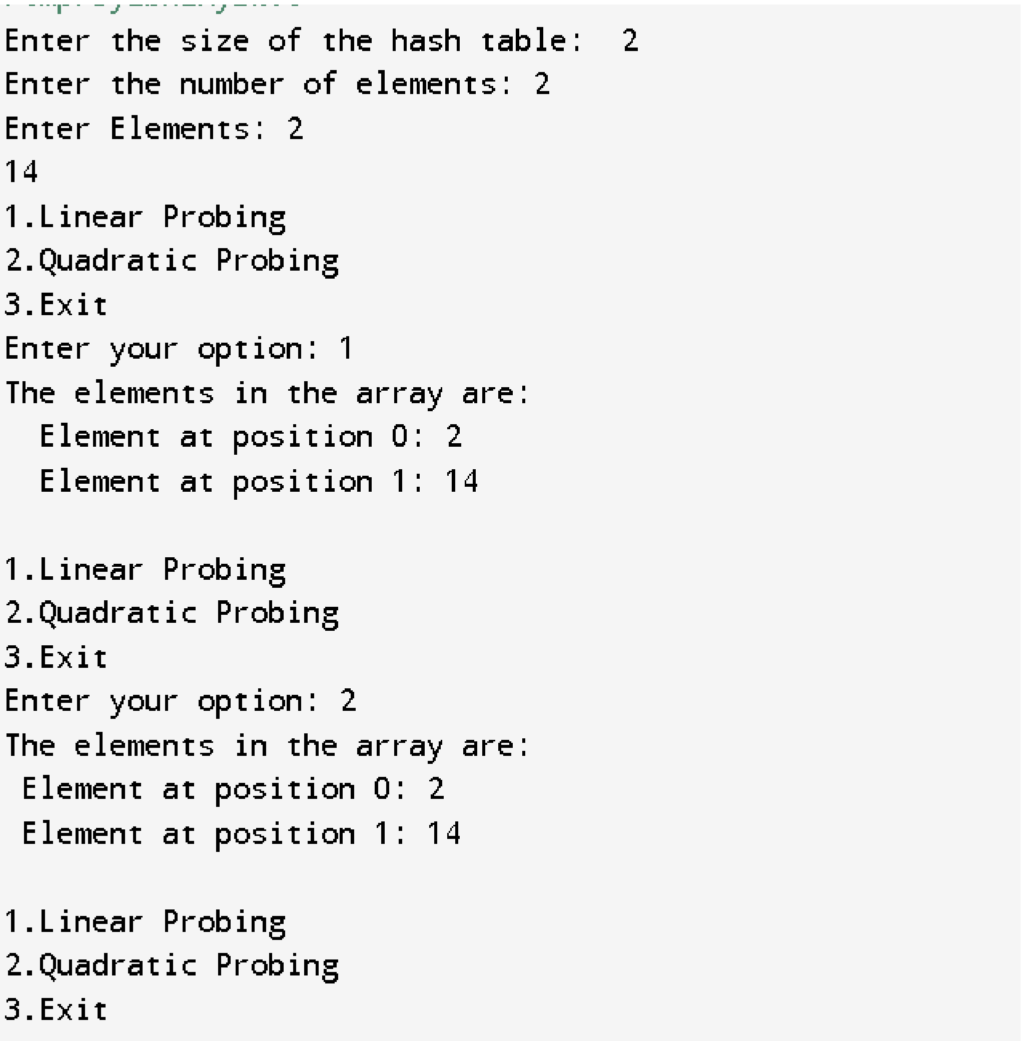


```
#include <stdio.h>
int tsize;
int hasht(int key)
{
 int i ;
 i = key%tsize ;
 return i;
}

//-------LINEAR PROBING-------
int rehashl(int key)
{
 int i ;
 i = (key+1)%tsize ;
 return i ;
}

//-------QUADRATIC PROBING-------
int rehashq(int key, int j)
{
 int i ;
 i = (key+(j*j))%tsize ;
 return i ;
}

void main()
{
    int key,arr[20],hash[20],i,n,s,op,j,k ;
    printf ("Enter the size of the hash table:  ");
    scanf ("%d",&tsize);

    printf ("\nEnter the number of elements: ");
    scanf ("%d",&n);

    for (i=0;i<tsize;i++)
 hash[i]=-1 ;

    printf ("Enter Elements: ");
    for (i=0;i<n;i++)
    {
 scanf("%d",&arr[i]);
    }
    do
    {
 printf("\n\n1.Linear Probing\n2.Quadratic Probing \n3.Exit \nEnter your
option: ");
 scanf("%d",&op);
 switch(op)
 {
 case 1:
     for (i=0;i<tsize;i++)
     hash[i]=-1 ;

     for(k=0;k<n;k++)
     {
  key=arr[k] ;
  i = hasht(key);
  while (hash[i]!=-1)
  {
      i = rehashl(i);
  }
  hash[i]=key ;
     }
     printf("\nThe elements in the array are: ");
     for (i=0;i<tsize;i++)
     {
  printf("\n  Element at position %d: %d",i,hash[i]);
     }
     break ;

 case 2:
     for (i=0;i<tsize;i++)
  hash[i]=-1 ;

     for(k=0;k<n;k++)
     {
  j=1;
  key=arr[k] ;
  i = hasht(key);
  while (hash[i]!=-1)
  {
      i = rehashq(i,j);
      j++ ;
  }
  hash[i]=key ;
     }
```



```
     printf("\nThe elements in the array are: ");
     for (i=0;i<tsize;i++)
     {
  printf("\n Element at position %d: %d",i,hash[i]);
     }
     break ;

 }
    }while(op!=3);
}
```

OUTPUT:



```
Enter the size of the hash table:  2
Enter the number of elements: 2
Enter Elements: 2
14
1.Linear Probing
2.Quadratic Probing
3.Exit
Enter your option: 1
The elements in the array are:
  Element at position 0: 2
  Element at position 1: 14

1.Linear Probing
2.Quadratic Probing
3.Exit
Enter your option: 2
The elements in the array are:
 Element at position 0: 2
 Element at position 1: 14

1.Linear Probing
2.Quadratic Probing
3.Exit
```

Conclusion: -Hence the concept of linear probing and quadratic probing was understood and encoded successfully.

Post lab questions-

a. **Compare and contrast various collision handling methods.**

Collision Handling Method	Pros	Cons
Linear Probing	- Simple to implement.	- Clustering: Can lead to clusters of filled slots, reducing performance.
	- Minimal extra memory usage.	- Performance degradation when the table is nearly full.
	- Easy to understand and code.	
Quadratic Hashing	- Mitigates clustering issues compared to linear probing.	- Can still exhibit clustering in certain cases.
	- Simple to implement like linear probing.	- May not provide as even distribution as more advanced methods.
Chaining	- Constant-time average-case performance for lookups, insertions, and deletions.	- Higher memory overhead due to linked lists.
	- Handles collisions well, even in high load scenarios.	- Slightly higher overhead in terms of space.

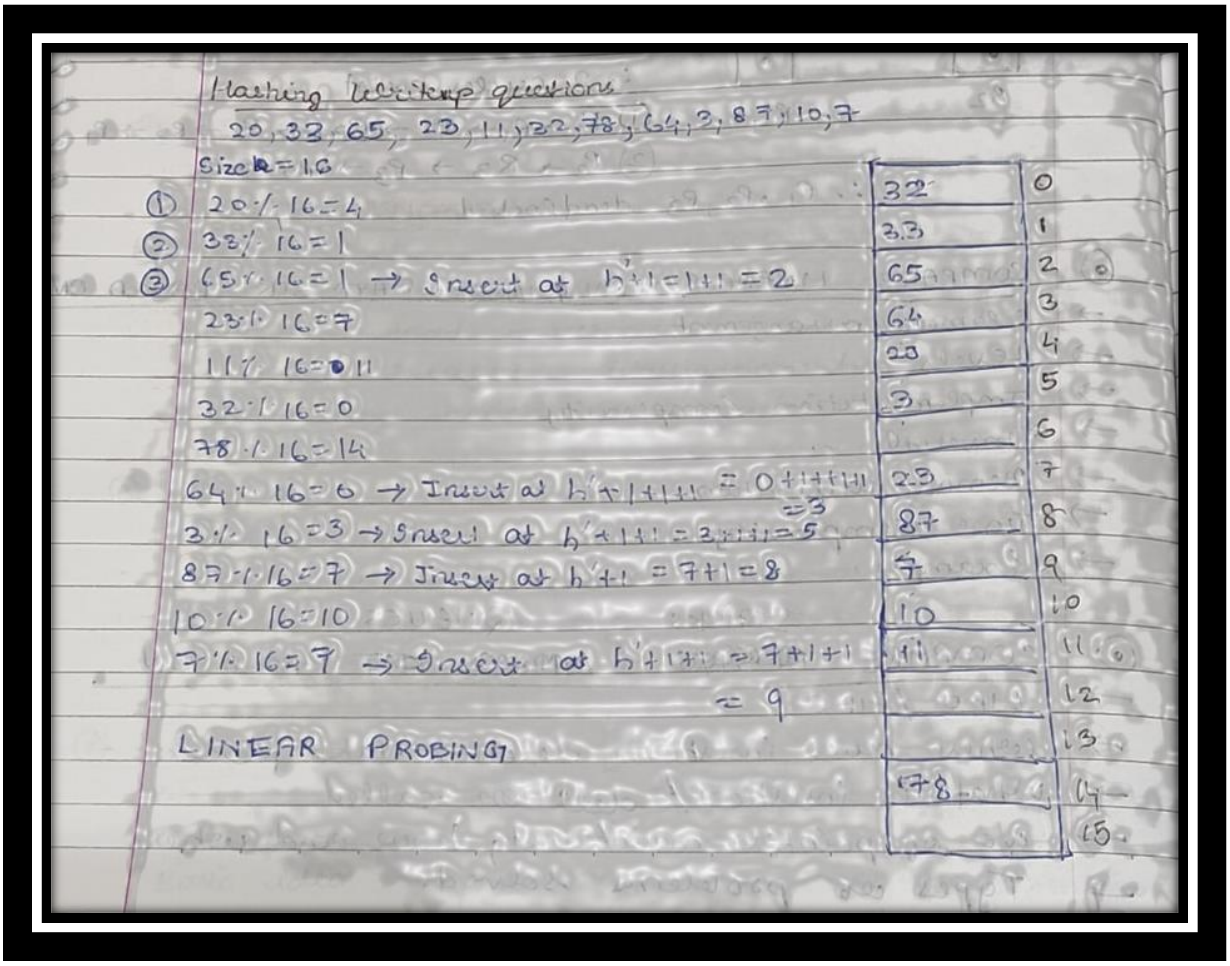
Collision Handling Method	Pros	Cons
	- No performance degradation as the table fills up.	

b. Store the given numbers in bucket of size 16, resolve the collisions if any with

  a. Linear probing
  b. Quadratic hashing
  c. Chaining

20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7

LINEAR PROBING



Hashing writeup questions

20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7

Size = 16

(1) 20 % 16 = 4

(2) 33 % 16 = 1

(3) 65 % 16 = 1 → Insert at h'+1 = 1+1 = 2

23 % 16 = 7

11 % 16 = 11

32 % 16 = 0

78 % 16 = 14

64 % 16 = 0 → Insert at h'+1+1+1 = 0+1+1+1 = 3

3 % 16 = 3 → Insert at h'+1+1 = 3+1+1 = 5

87 % 16 = 7 → Insert at h'+1 = 7+1 = 8

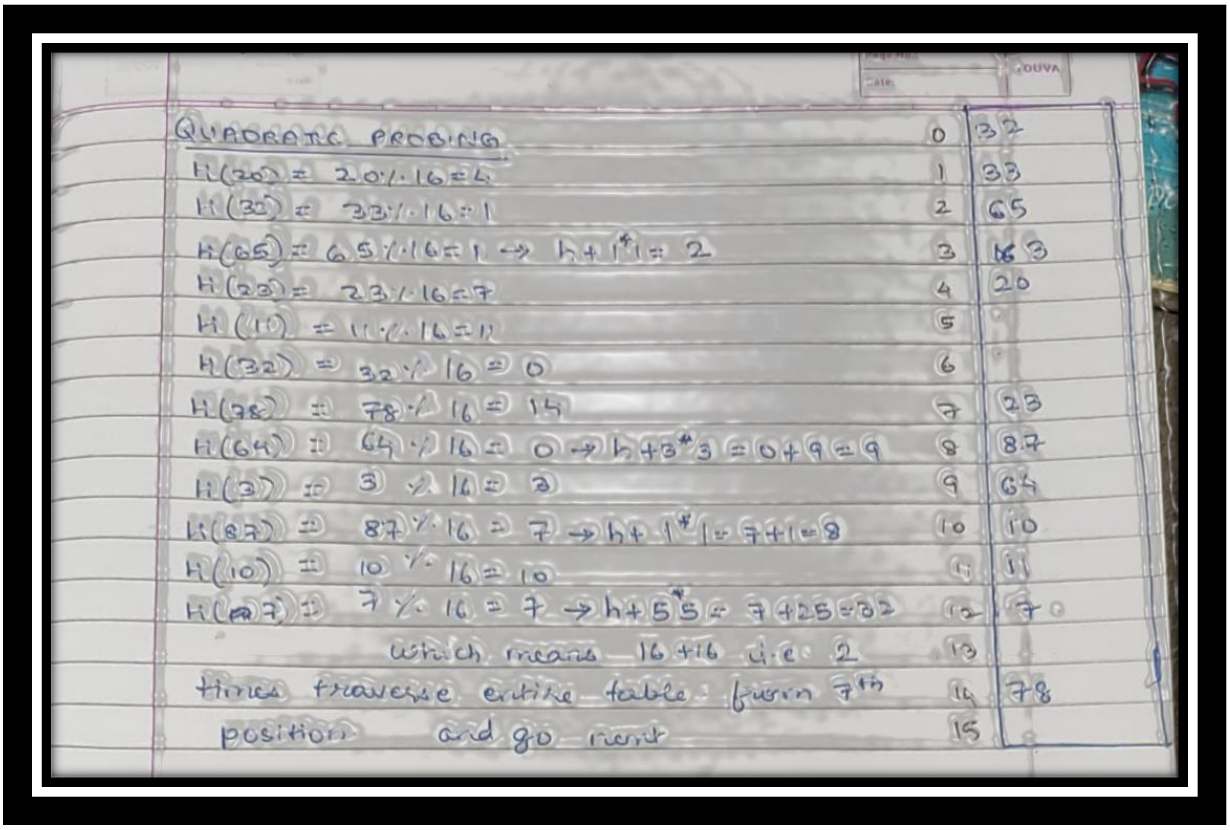
10 % 16 = 10

7 % 16 = 7 → Insert at h'+1+1 = 7+1+1 = 9

LINEAR PROBING

32	0
33	1
65	2
64	3
20	4
3	5
	6
23	7
87	8
7	9
10	10
11	11
	12
	13
78	14
	15

QUADRATIC PROBING

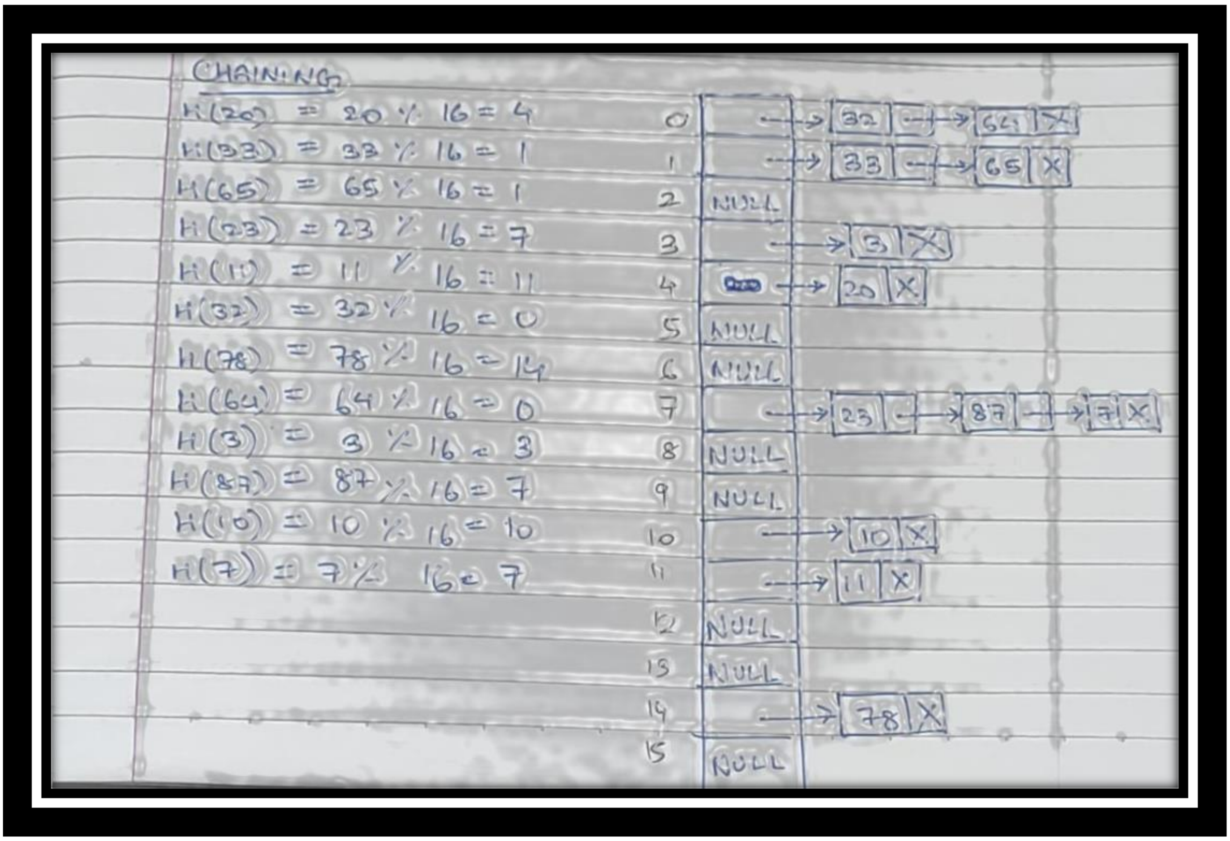


QUADRATIC PROBING

H(20) = 20 % 16 = 4
H(32) = 33 % 16 = 1
H(65) = 65 % 16 = 1 → h + 1*1 = 2
H(23) = 23 % 16 = 7
H(11) = 11 % 16 = 11
H(32) = 32 % 16 = 0
H(78) = 78 % 16 = 14
H(64) = 64 % 16 = 0 → h + 3*3 = 0 + 9 = 9
H(3) = 3 % 16 = 3
H(87) = 87 % 16 = 7 → h + 1*1 = 7 + 1 = 8
H(10) = 10 % 16 = 10
H(7) = 7 % 16 = 7 → h + 5*5 = 7 + 25 = 32
which means 16 + 16 i.e 2
times traverse entire table from 7th
position and go next

0	32
1	33
2	65
3	3
4	20
5	
6	
7	23
8	87
9	64
10	10
11	11
12	7
13	
14	78
15	

CHAINING



CHAINING

$H(20) = 20 \% 16 = 4$

$H(33) = 33 \% 16 = 1$

$H(65) = 65 \% 16 = 1$

$H(23) = 23 \% 16 = 7$

$H(11) = 11 \% 16 = 11$

$H(32) = 32 \% 16 = 0$

$H(78) = 78 \% 16 = 14$

$H(64) = 64 \% 16 = 0$

$H(3) = 3 \% 16 = 3$

$H(87) = 87 \% 16 = 7$

$H(10) = 10 \% 16 = 10$

$H(7) = 7 \% 16 = 7$