Hashing is the technique used for performing almost constant time search in case of insertion, deletion and find operation. Taking a very simple example of it, an array with its index as key is the example of hash table.
So each index (key) can be used for accessing the value in a constant search time. This mapping key must be simple to compute and must helping in identifying the associated value. Function which helps us in generating such kind of key-value mapping is known as **Hash Function.**

Hash Table a.k.a Hash Map is a data structure which uses hash function to generate key corresponding to the associated value.

lets look at some sample hash function for strings

Folding Method:-

```
int h(String x, int D)
{
    int i, sum;
    for (sum=0, i=0; i<x.length(); i++)
       sum+= (int)x.charAt(i);
    return (sum%D);
}
```

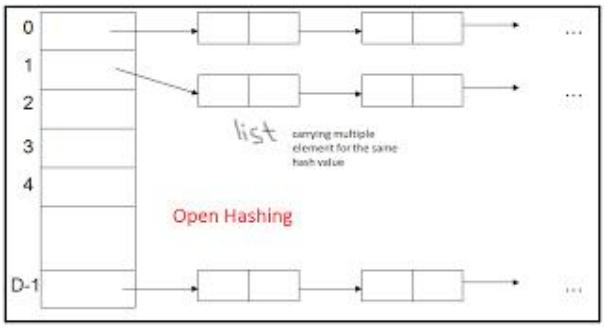
Cyclic Shift :-

```
static long hashCode(String key, int D)
{
 int h=0;
 for (int i=0, i<key.length(); i++)
 {
      h = (h << 4) | ( h >> 27);
      h += (int) key.charAt(i);
 }
 return h%D;
}
```

good link for hash function on string : click here

Coming to very important part of hashing , which is **collision resolution**. Since its always not possible to design perfect hash function with minimal overhead which would generate unique key. To address this problem following are the two main collision resolving techniques :-
1) Open Hashing also known as separate chaining
2) Closed Hashing also known as open addressing

Lets understand the difference between them
1) Open Hashing :- In this strategy collision is resolved by keeping the conflicting element in a list. That is to keep all element in a list which generate same hash.



Open Hashing

From above figure its clear that how collision get resolved by keeping a linked list.

2) Closed Hashing :- In this strategy collision is resolved by placing the conflicting element near to the slot generated by the hash function. Associated with closed hashing is a rehash strategy:
"If we try to place x in bucket h(x) and find it occupied, find alternative location h1(x), h2(x), etc. Try each in order, if none empty table is full,"
Lets take an example to understand it

HASH_TABLE_SIZE = 8
Input data :- a,b,c,d Hash for them H(a) = 0, H(b) = 3, H(c) = 7 and H(d) = 3

Now as 'c' and 'd' has same hash, where to insert 'd' then ?
Finding position using linear hashing :
h1(d) = (h(d)+1)%8 = 4%8 = 4

Adding 1 to hash function of h(d) we get new position 4, and slot 4 is currently non occupied. So entering d at position 4. In this way Closed hashing works.

Disadvantage of closed hashing is that it consumes more space as compared to open hashing
also it has less flexibility in accommodating for duplicate hash element.
Major *advantage* of closed hashing is that it reduces the overhead of introducing new data structure and reduces cost of new memory allocation per new element insertion.

Self Referencing Structure

Explain with an example the self-referential structure.

A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type. For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type. For example,

```
typedef struct listnode {
void *data;
struct listnode *next;
} linked_list;
```

In the above example, the listnode is a self-referential structure – because the *next is of the type sturct listnode.

A self referential structure is used to create data structures like linked lists, stacks, etc. Following is an example of this kind of structure:

```
struct struct_name
{
    datatype datatypename;
    struct_name * pointer_name;
};
```

```
/*
 * C Program for Interpolation search algorithm
 */
#include <stdio.h>
#define MAX 200

/* Inetrpolation search function */
int interpolation_search(int a[], int bottom, int top, int item)
{
        int mid;
        while (bottom <= top) {
                mid = bottom + (top - bottom) * ((item - a[bottom]) / (a[top] - a[bottom]));
                        if (item == a[mid])
                                return mid + 1;
                        if (item < a[mid])
                                top = mid - 1;
                        else
                                bottom = mid + 1;
        }
        return -1;
}
/* End of interpolation_search() */

/* The main() begins */
int main()
{
        int arr[MAX];
        int i, num;
        int item, pos;
        printf("\nEnter total elements (num< %d) : ", MAX);
        scanf("%d", &num);
        printf("Enter %d Elements in ascending order: ", num);
        for (i = 0; i < num; i++)
                        scanf("%d", &arr[i]);
        printf("\nSearch For : ");
        scanf("%d", &item);
        pos = interpolation_search(&arr[0], 0, num - 1, item);
        if (pos == -1)
                        printf("\nElement %d not found\n", item);
        else
                        printf("\nElement %d found at position %d\n", item, pos);
        return 0;
}
```