**Hashing**

Hashing is an efficient method to store and retrieve elements.

It’s exactly same as index page of a book. In index page, every topic is associated with a page number. If we want to look some topic, we can directly get the page number from the index.

Likewise, in hashing every value will be associated with a key. Using this key, we can point out the element directly.

Let’s discuss hashing with modulo method.

How to calculate the hash key?

Let's take hash table size as 7.

size = 7

arr[size];

**Formula to calculate key is,**

**key = element % size**

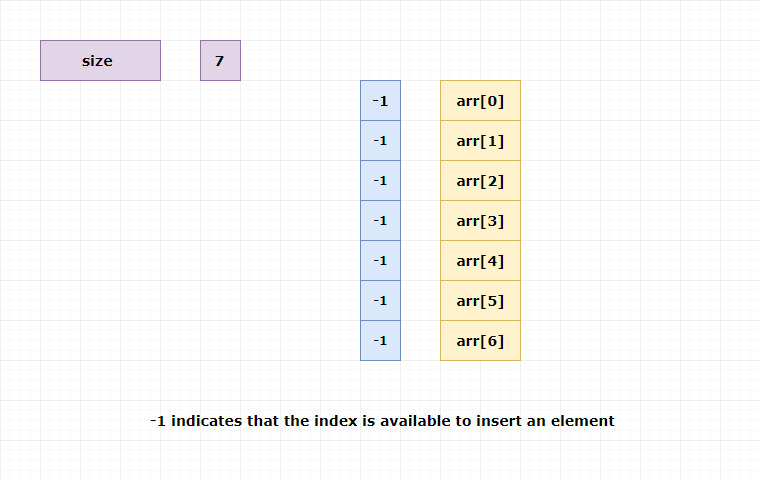
If we take modulo of number with N, the remainder will always be 0 to N - 1.

Exactly array index also starts from 0 and ends with index N -1. So we can easily store elements in array index.

Initialize the Hash Bucket

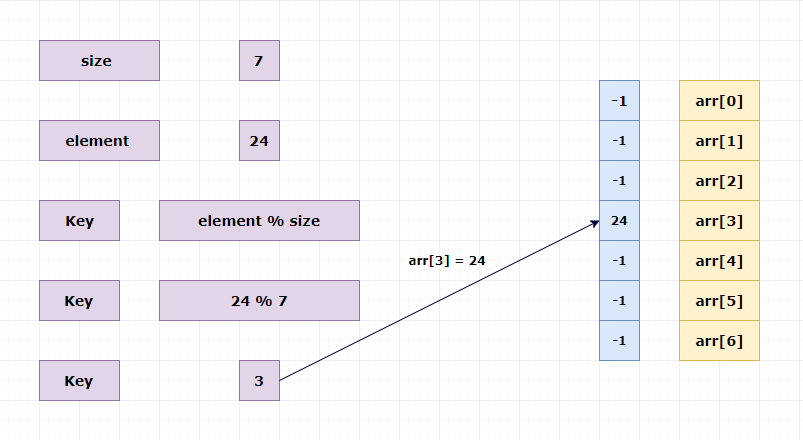
Before inserting elements into array. Let’s make array default value as -1.

-1 indicates element not present or the particular index is available to insert.

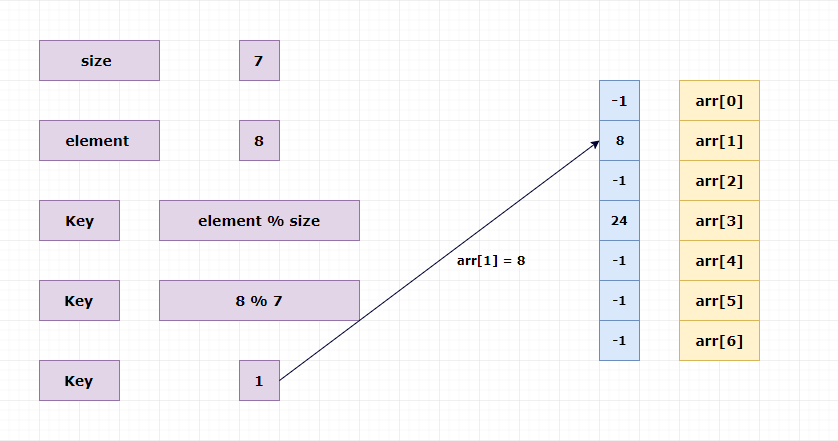


Inserting elements in the hash table

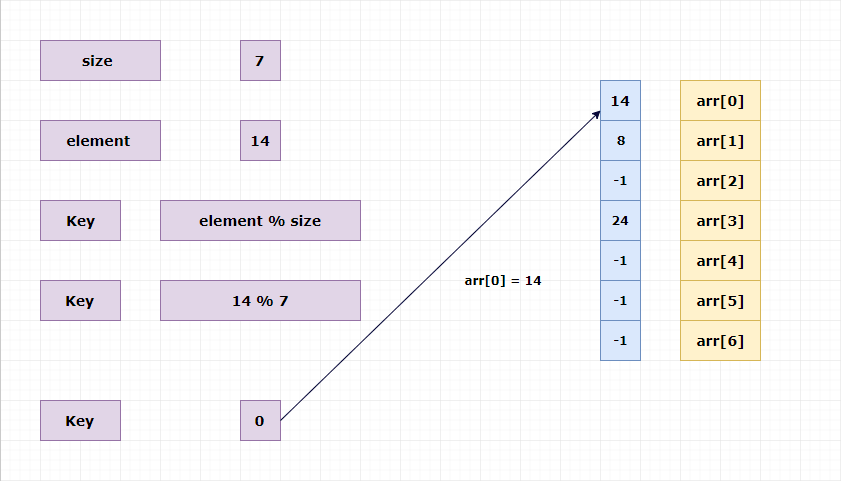
i)insert 24



ii)insert 8

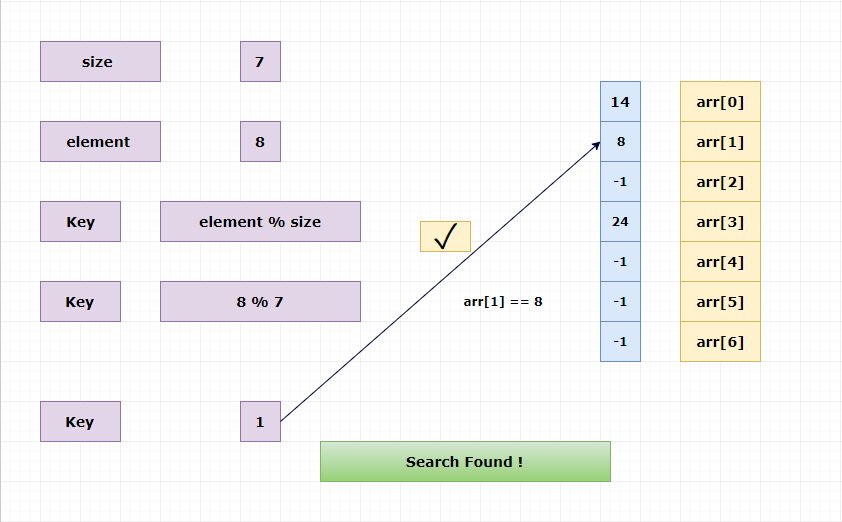


iii)insert 14

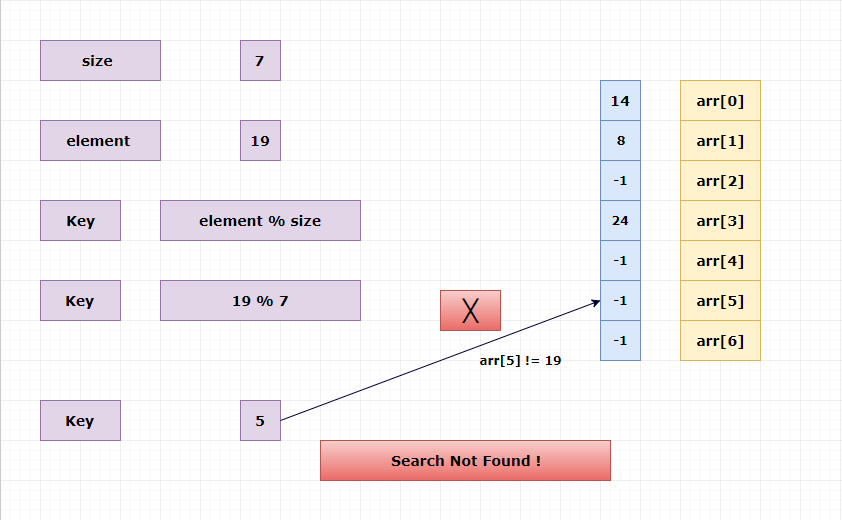


Searching elements from the hash table

i)search 8



ii)search 19



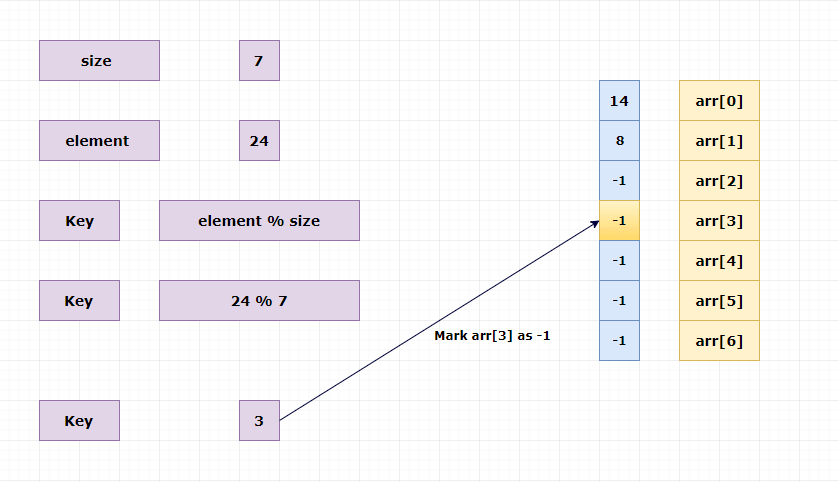
Deleting an element from the hash table

Here, we are not going to remove the element.

We just mark the index as -1. It is indirectly delete the element from array.

**Example**

Delete: 24



What is collision in hashing?

What if we insert an element say 15 to existing hash table?

Insert : 15

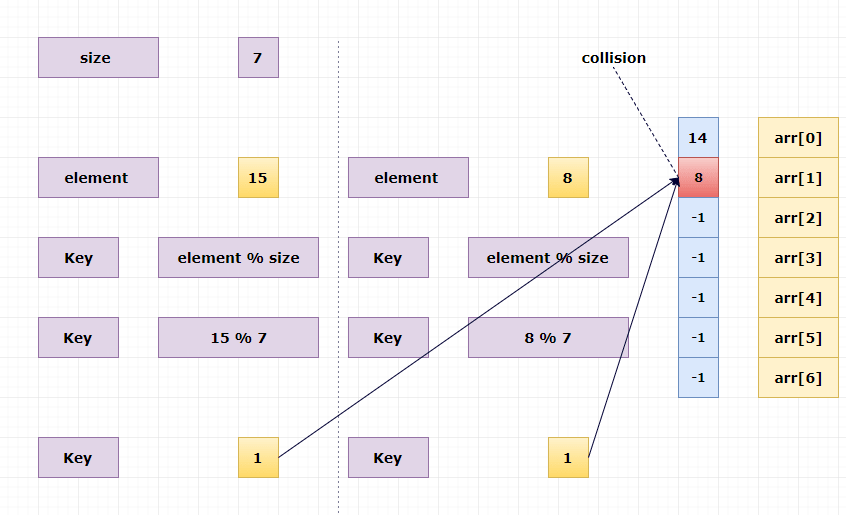
Key = element % key

Key = 15 % 7

Key = 1

But already arr[1] has element 8 !

Here, two or more different elements pointing to the same index under modulo size. This is called **collision**.



Hash table implementation in c using arrays

Example

#include<stdio.h>

#define size 7

**int** arr[size];

**void** **init**()

{

**int** i;

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

arr[i] = -**1**;

}

**void** **insert**(**int** value)

{

**int** key = value % size;

**if**(arr[key] == -**1**)

{

arr[key] = value;

printf("%d inserted at arr[%d]**\n**", value,key);

}

**else**

{

printf("Collision : arr[%d] has element %d already!**\n**",key,arr[key]);

printf("Unable to insert %d**\n**",value);

}

}

**void** **del**(**int** value)

{

**int** key = value % size;

**if**(arr[key] == value)

arr[key] = -**1**;

**else**

printf("%d not present in the hash table**\n**",value);

}

**void** **search**(**int** value)

{

**int** key = value % size;

**if**(arr[key] == value)

printf("Search Found**\n**");

**else**

printf("Search Not Found**\n**");

}

**void** **print**()

{

**int** i;

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

printf("arr[%d] = %d**\n**",i,arr[i]);

}

**int** **main**()

{

init();

insert(**10**); //key = 10 % 7 ==> 3

insert(**4**); //key = 4 % 7 ==> 4

insert(**2**); //key = 2 % 7 ==> 2

insert(**3**); //key = 3 % 7 ==> 3 (collision)

printf("Hash table**\n**");

print();

printf("**\n**");

printf("Deleting value 10..**\n**");

del(**10**);

printf("After the deletion hash table**\n**");

print();

printf("**\n**");

printf("Deleting value 5..**\n**");

del(**5**);

printf("After the deletion hash table**\n**");

print();

printf("**\n**");

printf("Searching value 4..**\n**");

search(**4**);

printf("Searching value 10..**\n**");

search(**10**);

**return** **0**;

}

[Run it](https://www.log2base2.com/algorithms/searching/try-it-hashing-in-c-data-structure.html)

We didn't implement any collision avoidance technique in the above code.

We will discuss collision avoidance in the next tutorials.