

**Batch: A3 Roll No.:16010421075 Experiment No.: 7 Aim:** Write a menu driven program to implement a hash table.

**Resources Used:** C/ C++ editor and compiler.



# Theory:

**Hash Table**

Hash table is one of the most important data structures that uses a special function known as a hash function that maps a given value with a key to access the elements faster.

A Hash table is a data structure that stores some information, and the information has basically two main components, i.e., key and value. The hash table can be implemented with the help of an associative array. The efficiency of mapping depends upon the efficiency of the hash function used for mapping.

For example, suppose the key value is John and the value is the phone number, so when we pass the key value in the hash function shown as below:

Hash(key)= index;

When we pass the key in the hash function, then it gives the index. Hash(john) = 3;

The above example adds the john at the index 3.

# Hashing

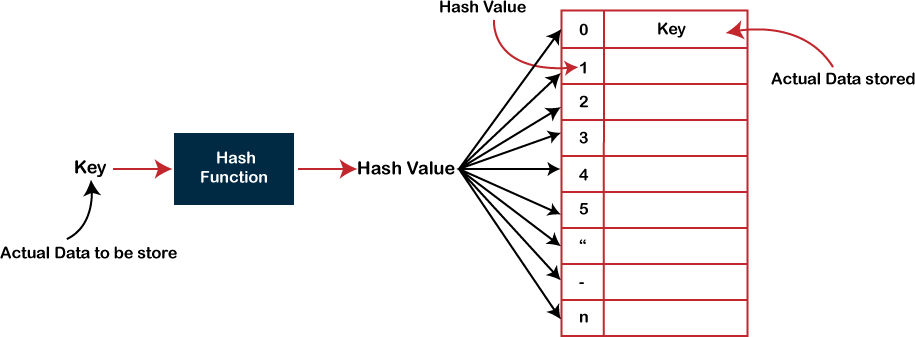
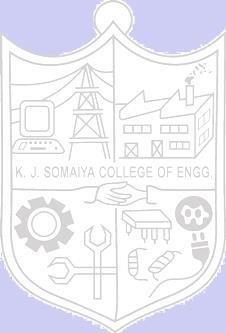
Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1).

The worst time complexity in linear search is O(n), and O(logn) in binary search. In both the searching techniques, the searching depends upon the number of elements but we want the technique that takes a constant time. So, hashing technique came that provides a constant time.

In Hashing technique, the hash table and hash function are used. Using the hash function, we can calculate the address at which the value can be stored.

The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:

# Index = hash(key)



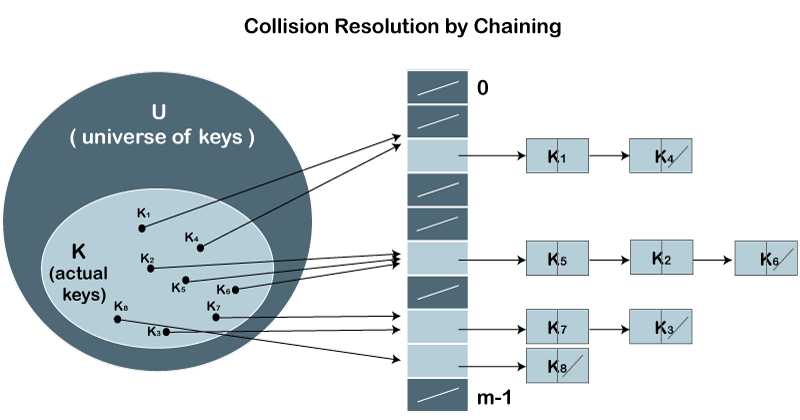
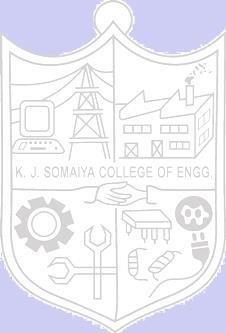
Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

The following are the collision techniques:

* Open Hashing: It is also known as closed addressing.
* Closed Hashing: It is also known as open addressing.

# Open Hashing

* In Open Hashing, one of the methods used to resolve the collision is known as a chaining method.



Define a data item having some data and key, based on which the search is to be conducted in a hash table.

struct DataItem { int data;

int key;

};

# Hash Method

Define a hashing method to compute the hash code of the key of the data item.



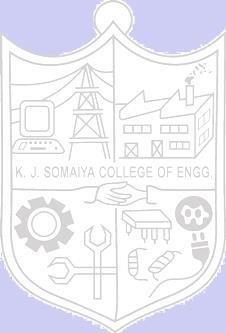
# Search Operation

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.

# Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.

# Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact.

Activity: A C program depicting the searching and insertion operation in a hash table.

# Results: CODE:

#include <stdio.h> #include <stdlib.h>

struct set

{

int key; int data;

};

struct set \*array; int capacity = 10; int size = 0;

int hashFunction(int key)

{

return (key % capacity);

}

int checkPrime(int n)

{

int i;

if (n == 1 || n == 0)

{

return 0;

}

for (i = 2; i < n / 2; i++)

{

if (n % i == 0)

{

return 0;

}

}

return 1;

}

int getPrime(int n)

{

if (n % 2 == 0)

{ n++;

}

while (!checkPrime(n))

{

n += 2;

}

return n;

}

void init\_array()

{

capacity = getPrime(capacity);

array = (struct set \*)malloc(capacity \* sizeof(struct set)); for (int i = 0; i < capacity; i++)

{

array[i].key = 0;

array[i].data = 0;

}

}

void insert(int key, int data)

{

int index = hashFunction(key); if (array[index].data == 0)

{

array[index].key = key; array[index].data = data; size++;

printf("\n Key (%d) has been inserted \n", key);

}

else if (array[index].key == key)

{

array[index].data = data;

}

else

{

printf("\n Collision occured \n");

}

}

void remove\_element(int key)

{

int index = hashFunction(key); if (array[index].data == 0)

{

printf("\n This key does not exist \n");

}

else

{

array[index].key = 0;

array[index].data = 0; size--;

printf("\n Key (%d) has been removed \n", key);

}

}

void display()

{

int i;

for (i = 0; i < capacity; i++)

{

if (array[i].data == 0)

{

printf("\n array[%d]: / ", i);

}

else

{

printf("\n key: %d array[%d]: %d \t", array[i].key, i, array[i].data);

}

}

}

void search(int key)

{

int index = hashFunction(key); if (array[index].data == 0)

{

printf("\n This key does not exist \n");

}

else

printf("\n Key (%d) is present \n", key);

}

int size\_of\_hashtable()

{

return size;

}

int main()

{

int choice, key, data, n; int c = 0;

init\_array();

do

{

printf("1.Insert item in the Hash Table" "\n2.Remove item from the Hash Table" "\n3.Check the size of Hash Table" "\n4.Display a Hash Table"

"\n5.Search an item from the Hash table" "\n\n Please enter your choice: ");

scanf("%d", &choice); switch (choice)

{

case 1:

printf("Enter key -:\t");

scanf("%d", &key); printf("Enter data -:\t"); scanf("%d", &data); insert(key, data);

break; case 2:

printf("Enter the key to delete-:"); scanf("%d", &key); remove\_element(key);

break; case 3:

n = size\_of\_hashtable();

printf("Size of Hash Table is-:%d\n", n); break;

case 4:

display(); break; case 5:

printf("Enter the key to search-:"); scanf("%d", &key);

search(key); break; default:

printf("Invalid Input\n");

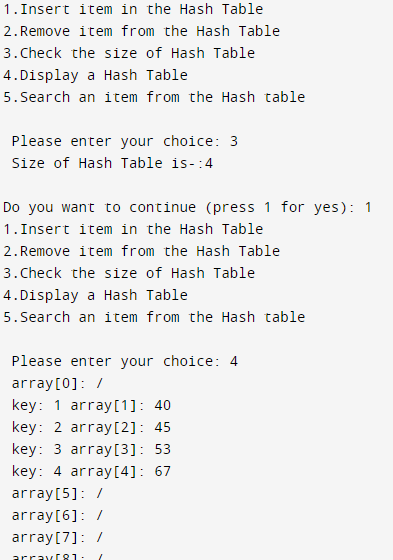
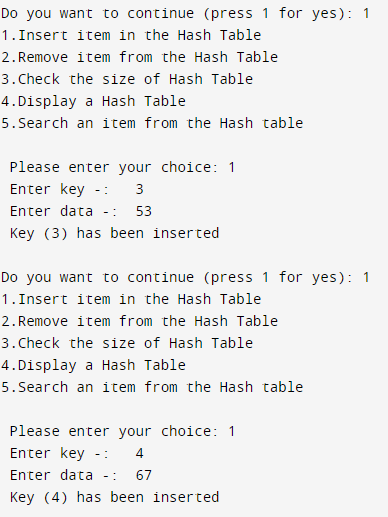
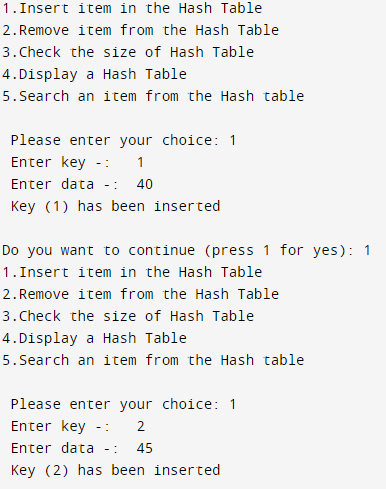
}

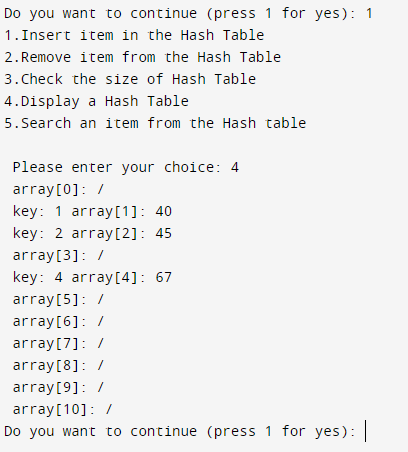
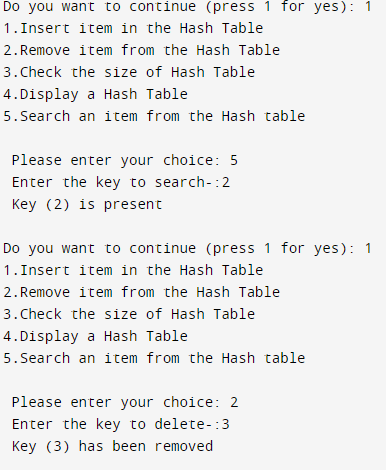
printf("\nDo you want to continue (press 1 for yes): "); scanf("%d", &c);

} while (c == 1);

}

**OUTPUT:**







**Outcomes:** CO3: Describe concepts of advance data structures like set, map & dictionary.



**Conclusion:** Therefore we wrote a menu driven program to implement a hash table.

# Grade: AA / AB / BB / BC / CC / CD /DD Signature of faculty in-charge with date

**References:**

# Books/ Journals/ Websites:

* + Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing.