# Assignment 7

## Aim:

Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod (m-1)).

## Objective:

To understand :

1. How keys can be mapped to the corresponding values , in a hash table, in order to have the lowest time complexity.

2. How collisions can be resolved , in a hash table , using a second hash function.

## Theory:

Double hashing is a computer programming technique , used in hash tables to resolve hash collisions, in cases when two different values to be searched for produce the same hash key. It is a popular collision -resolution technique in open-addressed hash tables. Double hashing is implemented in many popular libraries.

Like linear probing, it uses one hash value as a starting point and then repeatedly steps forward an interval until the desired value is located, an empty location is reached, or the entire table has been searched; but this interval is decided using a second, independent hash function (hence the name double hashing). Unlike linear probing and quadratic probing , the interval depends on the data, so that even values mapping to the same location have different bucket sequences; this minimizes repeated collisions and the effects of clustering.

First hash function is typically hash1(key) = key % TABLE_SIZE

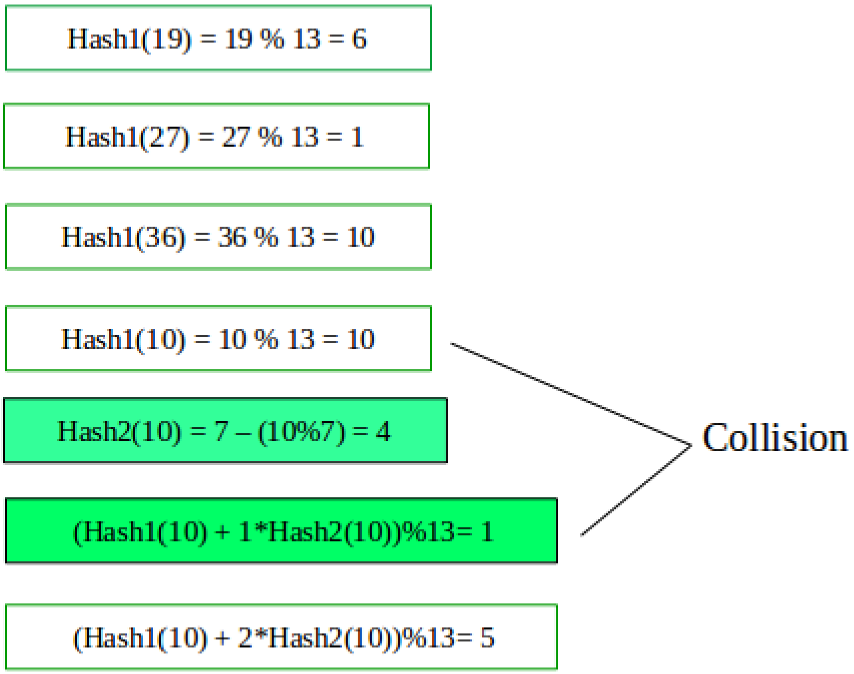
A popular second hash function is : **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.
A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed

**Example:**

**Lets say, Hash1 (key) = key % 13**

**Hash2 (key) = 7 – (key % 7)**



Hash1(19) = 19 % 13 = 6

Hash1(27) = 27 % 13 = 1

Hash1(36) = 36 % 13 = 10

Hash1(10) = 10 % 13 = 10

Hash2(10) = 7 – (10%7) = 4

(Hash1(10) + 1*Hash2(10))%13= 1

(Hash1(10) + 2*Hash2(10))%13= 5

# Algorithm:

1. Start.
2. Accept the size of the table.
3. Initialize the hash table array to any negative integer value say "-111"(Provided negative keys are not accepted in the table).
4. Map the key to it's value, using first hash function: hash1(key) = key % Table_size.
5. If collision occurs use the second hash function: hash2(key) = 1+(key mod (size-1)).
6. Do: Hi(key)=((Hash(key) + i * hash2(key)) mod size) , using a for loop, for i from 1 to (size-1), untill the key gets mapped to it's appropriate value.
7. Stop.

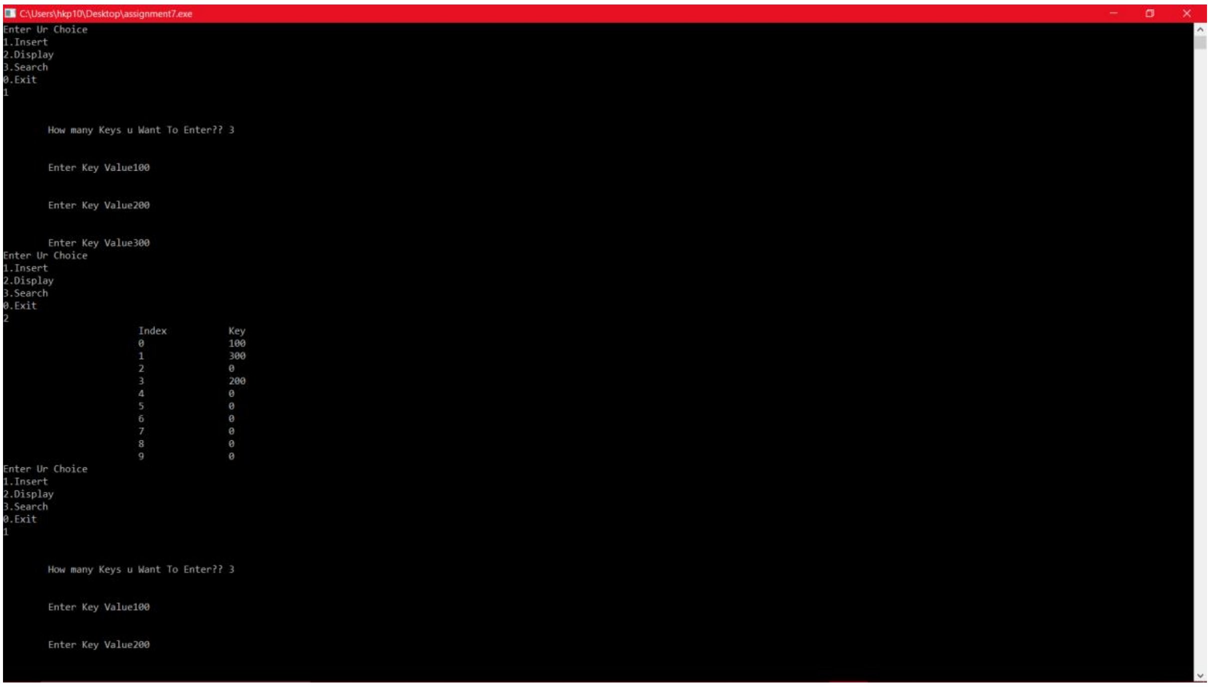
# Code:

```
#include<iostream>
using namespace std;



class hashTable
{


public:
        int data[10],occ[10];
        int key,index=0,index2=0,n;


hashTable()
{


        for(int i=0;i<10;i++)
        {
                occ[i]=0;
                data[i]=0;
        }




}


void insert();
void calIndex();
void display();
void search();
void delet();
};



void hashTable::insert()
{

        cout<<"\n\n\tHow many Keys u Want To Enter?? ";
        cin>>n;

for(int i=0;i<n;i++)
{
        cout<<"\n\n\tEnter Key Value";
        cin>>key;

        index = (key % 10);
        calIndex();
}

}



void hashTable::calIndex()
```

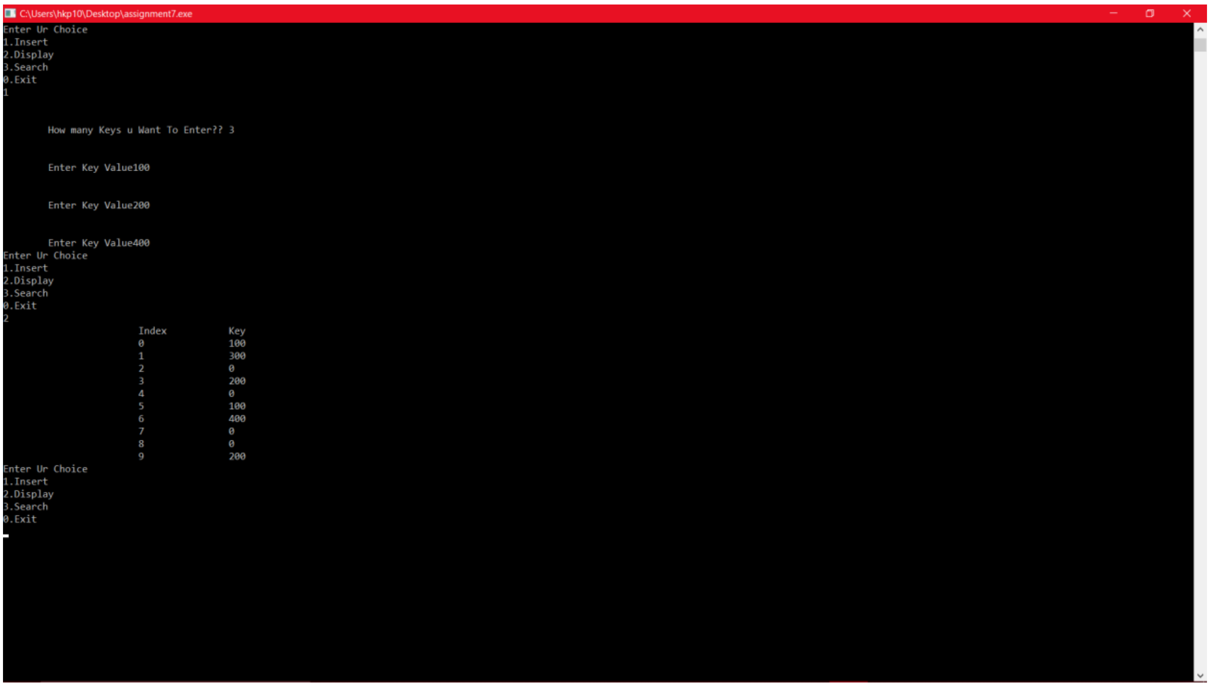
```
{

        if(occ[index]==0)
        {
        data[index] = key;
        occ[index] = 1;
        }
        else if(occ[index] == 1)
        {
        for(int j=0;j<10;j++)
        {
                index2 = 7 - (key % 7);

                index = (index + j*index2)%10;

                if(occ[index] == 0)
                        break;
        }
        data[index] = key;
        occ[index] = 1;

        }
}




void hashTable::display()
```

```
{
        cout<<"\t\t\tIndex "<<"\t\tKey\n";
        for(int i=0;i<10;i++)
        cout<<"\t\t\t"<<i<<"\t\t"<<data[i]<<"\n";

}



void hashTable::search()
{
        int search;
        cout<<"\n\n\tEnter Key to be Searched ";
        cin>>search;

        for(int i=0;i<10;i++)
        {
        if(data[i]==search)
        {
                cout<<"\n\t\t"<<search<<" Found at Index "<<i<<"\n";
        }
        }
}



int main()
{
        int ch;
        hashTable h1;

        do{

        cout<<"Enter Ur Choice\n1.Insert\n2.Display\n3.Search\n0.Exit\n";
        cin>>ch;

        switch(ch)
        {
                case 1: h1.insert();
                                break;
                case 2: h1.display();
                                break;
                case 3: h1.search();
                                break;

        }
}while(ch!=0);

}
```

# OUTPUT:



```
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
1


        How many Keys u Want To Enter?? 3


        Enter Key Value100


        Enter Key Value200


        Enter Key Value300
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
2
                        Index           Key
                        0               100
                        1               300
                        2               0
                        3               200
                        4               0
                        5               0
                        6               0
                        7               0
                        8               0
                        9               0
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
1


        How many Keys u Want To Enter?? 3


        Enter Key Value100


        Enter Key Value200
```



```
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
1


        How many Keys u Want To Enter?? 3


        Enter Key Value100


        Enter Key Value200


        Enter Key Value400
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
2
                        Index           Key
                        0               100
                        1               300
                        2               0
                        3               200
                        4               0
                        5               100
                        6               400
                        7               0
                        8               0
                        9               200
Enter Ur Choice
1.Insert
2.Display
3.Search
0.Exit
_
```

# Conclusion:

Hence Double Hashing can be used in this way to solve problem of collision.