Problem Statement: Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations :Insert(key, value), Find(key),Delete(key).

```
n = int(input("Enter limit of database : "))
values = int(input("Enter values you will enter : "))

hashTable=[[-1,-1,-1]]*n
def hash_function(key):
    return key%n

def display_list(list):
    for i in range(n):
        print(i," -> ",list[i])

def insert_key(k,v):
    addr=hash_function(k)
    if hashTable[addr][0]==-1:
        temp=[]
        temp.append(k)
        temp.append(v)
        temp.append(-1)
        hashTable.pop(addr)
        hashTable.insert(addr,temp)
    else:
        i=addr
        while hashTable[i][0]!=-1:
        i+=1
        if i>=n:
        i=0
        temp=[]
        temp.append(k)
        temp.append(v)
        temp.append(-1)
        hashTable.pop(i)
        hashTable.insert(i,temp)
        print(hashTable,"*******")
        pos=i
        i=addr
        index=[]
        while hashTable[i][0]!=-1:
        if hash_function(hashTable[i][0])==addr:
```

```
                index.append(i)
                print(index,"########")
        i+=1
        if i>=n:
        i=0
        i=1
        for j in index:
        if i>=(len(index)):
                break
        else:
                hashTable[j][2]=index[i]
                i+=1
        print(hashTable)

def search_key(key):
        s=-1
        for i in range(n):
        if hashTable[i][0] == key:
        s=i
        break
        return s
def delete_key(key):
        pos=search_key(key)
        print(pos)
        if pos==-1:
        return -1
        else:
        temp=hashTable[pos]
        print(temp)
        #if temp[2]!=-1:
        addr=hash_function(key)
        chain=[]
        i=0
        while i<n:
        if hash_function(hashTable[i][0])==addr:
                t=[]
                t.append(i)
                t.append(hashTable[i][0])
                t.append(hashTable[i][2])
                chain.append(t)
        i+=1
        print(chain)
        for j in range(len(chain)):
        if chain[j][2]==pos
```

```
                chain[j][2]=hashTable[pos][2]
                break
        #hashTable[]
        p=chain[j][0]
        hashTable[p][2]=chain[j][2]
        hashTable[pos][0]=-1
        hashTable[pos][1]=-1
        hashTable[pos][2]=-1
        print("Key Deleted")

'''def Chaining_With_Replacement(k,v):
    addr=hash_function(k)
    index=[]
    if hashTable[addr][0]==-1:
        temp=[]
        temp.append(k)
        temp.append(v)
        temp.append(-1)
        hashTable.pop(addr)
        hashTable.insert(addr,temp)
    else:
        i=addr
        if hash_function(hashTable[i][0])==addr:
        while hashTable[i][0]!=-1:
            i+=1
            if i>=n:
            i=0
        temp=[]
        temp.append(k)
        temp.append(v)
        temp.append(-1)
        hashTable.pop(i)
        hashTable.insert(i,temp)
        print(hashTable,"*******")
        pos=i
        i=addr
        #index=[]
        while hashTable[i][0]!=-1:
            if hash_function(hashTable[i][0])==addr:
            index.append(i)
            print(index,"########")
            i+=1
            if i>=n:
            i=1
```

```
        for j in index:
                hashTable[j][2]=index[i]
                i+=1
                if i>=(len(index)):
                break
        print(hashTable)
        else:
        temp1=hashTable[addr]
        temp=[]
        temp.append(k)
        temp.append(v)
        temp.append(-1)
        hashTable.pop(addr)
        hashTable.insert(addr,temp)
        i=addr
        while hashTable[i][0]!=-1:
                i+=1
                if i>=n:
                i=0
        hashTable.pop(i)
        hashTable.insert(i,temp1)
        print(temp1,"&&&&&&&&&")
        pos=addr
        new_chain=i
        addr=hash_function(temp1[0])
        i=addr
        while hashTable[i][0]!=-1:
                if hashTable[i][2]==pos:
                hashTable[i][2]=new_chain
                break
                i+=1
                if i>=n:
                i=0'"
print(hashTable)

for i in range(values):
    key = int(input("Enter key : "))
    value = int(input("Enter value : "))
    insert_key(key,value)
    #Chaining_With_Replacement(key,value)
#print(hashTable)
display_list(hashTable)
#the element at that position
#display_list(list)
```

```
searchKey = int(input("\nEnter key to be searched : "))
s=search_key(searchKey)
if s!=-1:
        print("\n***Found at position ",s,"****")
else:
        print("\n***Key not Found ****")

deleteKey = int(input("\nEnter key to be deleted : "))
delete_key(deleteKey)
display_list(hashTable)
```

OUTPUT:

```
Enter limit of database : 6
Enter values you will enter : 5
[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1], [-1, -1, -1], [-1, -1, -1], [-1, -1, -1]]
Enter key : 23
Enter value : 1
Enter key : 33
Enter value : 2
Enter key : 45
Enter value : 3
[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1], [33, 2, -1], [45, 3, -1], [23, 1, -1]]
    *******
[3] ########
[3, 4] ########
[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1], [33, 2, 4], [45, 3, -1], [23, 1, -1]]
Enter key : 78
Enter value : 4
Enter key : 98
Enter value : 5
0  ->  [78, 4, -1]
1  ->  [-1, -1, -1]
2  ->  [98, 5, -1]
3  ->  [33, 2, 4]
4  ->  [45, 3, -1]
5  ->  [23, 1, -1]
```

```
Enter key to be deleted : 98
2
[98, 5, -1]
[[2, 98, -1]]
Key Deleted
0   ->   [78, 4, -1]
1   ->   [-1, -1, -1]
2   ->   [-1, -1, -1]
3   ->   [33, 2, 4]
4   ->   [45, 3, -1]
5   ->   [23, 1, -1]
```