Assignment no.: 6

Name : Kapare Swapnali Namdev

Class : SE

Div : A

Roll no : A- 65

Problem statement :

Design and implement a hash table of fixed size. Use the division method for the hash function and resolve collisions using linear probing. Allow the user to perform the following operations:

- Insert a key
- Search for a key
- Delete a key
- Display the table

Input :

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [None] * size
        self.deleted = "<deleted>"

    def _hash_function(self, key):
```

```
    return key % self.size

def insert(self, key):
    index = self._hash_function(key)
    start_index = index
    while self.table[index] not in (None, self.deleted):
        if self.table[index] == key:
            print(f"Key {key} already exists at index {index}")
            return
        index = (index + 1) % self.size
        if index == start_index:
            print("Hash table is full. Cannot insert.")
            return
    self.table[index] = key
    print(f"Inserted key {key} at index {index}")

def search(self, key):
    index = self._hash_function(key)
    start_index = index
    while self.table[index] is not None:
        if self.table[index] == key:
            print(f"Key {key} found at index {index}")
            return index
        index = (index + 1) % self.size
        if index == start_index:
```

```
            break
        print(f"Key {key} not found")
        return None

    def delete(self, key):
        index = self._hash_function(key)
        start_index = index
        while self.table[index] is not None:
            if self.table[index] == key:
                self.table[index] = self.deleted
                print(f"Key {key} deleted from index {index}")
                return
            index = (index + 1) % self.size
            if index == start_index:
                break
        print(f"Key {key} not found for deletion")

    def display(self):
        print("Hash Table Contents:")
        for i, val in enumerate(self.table):
            print(f"Index {i}: {val}")

# 🔧 Demonstration
ht = HashTable()
```

```
# Insert keys
ht.insert(10)
ht.insert(20)
ht.insert(30)
ht.insert(25)  # Will cause collision with 25 % 10 = 5

# Search keys
ht.search(10)
ht.search(25)
ht.search(99)  # Not present

# Delete keys
ht.delete(20)
ht.delete(99)  # Not present

# Display final state
ht.display()
```

Output :

```
Inserted key 10 at index 0
Inserted key 20 at index 1
Inserted key 30 at index 2
Inserted key 25 at index 5
Key 10 found at index 0
Key 25 found at index 5
```

Key 99 not found

Key 20 deleted from index 1

Key 99 not found for deletion

Hash Table Contents:

Index 0: 10

Index 1: <deleted>

Index 2: 30

Index 3: None

Index 4: None

Index 5: 25

Index 6: None

Index 7: None

Index 8: None

Index 9: None