Practical No 10

Aim: Write A Program To Implement Hashing Concepts & Collision Handling

```
# -------------------- Hash Table with Chaining --------------------
class HashTableChaining:
    def __init__(self, size=7):
        self.size = size
        self.table = [[] for _ in range(size)]

    def hash_function(self, key):
        return key % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        # Update if key already exists
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])

    def search(self, key):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None

    def delete(self, key):
        index = self.hash_function(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                del self.table[index][i]
                return True
        return False

    def display(self):
        print("\nHash Table (Chaining):")
        for i, bucket in enumerate(self.table):
            print(i, "->", bucket)

# -------------------- Hash Table with Linear Probing --------------------
class HashTableLinearProbing:
    def __init__(self, size=7):
        self.size = size
        self.table = [None] * size

    def hash_function(self, key):
        return key % self.size
```

```
    def insert(self, key, value):
        index = self.hash_function(key)
        start_index = index
        while self.table[index] is not None and self.table[index][0] != key:
            index = (index + 1) % self.size
            if index == start_index:
                print("Hash Table Full! Cannot insert:", key)
                return
        self.table[index] = (key, value)

    def search(self, key):
        index = self.hash_function(key)
        start_index = index
        while self.table[index] is not None:
            if self.table[index][0] == key:
                return self.table[index][1]
            index = (index + 1) % self.size
            if index == start_index:
                break
        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][0] == key:
                self.table[index] = None
                return True
            index = (index + 1) % self.size
            if index == start_index:
                break
        return False

    def display(self):
        print("\nHash Table (Linear Probing):")
        for i, val in enumerate(self.table):
            print(i, "->", val)

# -------------------- Example Usage --------------------
if __name__ == "__main__":
    # Using Chaining
    ht_chain = HashTableChaining()
    ht_chain.insert(10, "Alice")
    ht_chain.insert(20, "Bob")
    ht_chain.insert(30, "Charlie")
    ht_chain.insert(17, "David")  # Collides with 10 (10 % 7 == 3, 17 % 7 == 3)
    ht_chain.display()
    print("Search key 20:", ht_chain.search(20))
    ht_chain.delete(10)
    ht_chain.display()

    # Using Linear Probing
```

```
    ht_lp = HashTableLinearProbing()
    ht_lp.insert(10, "Red")
    ht_lp.insert(20, "Blue")
    ht_lp.insert(30, "Green")
    ht_lp.insert(17, "Yellow")  # Collision, will probe next slot
    ht_lp.display()
    print("Search key 17:", ht_lp.search(17))
    ht_lp.delete(20)
    ht_lp.display()
```

```
Hash Table (Chaining):
0 -> []
1 -> []
2 -> [[30, 'Charlie']]
3 -> [[10, 'Alice'], [17, 'David']]
4 -> []
5 -> []
6 -> [[20, 'Bob']]
Search key 20: Bob

Hash Table (Chaining):
0 -> []
1 -> []
2 -> [[30, 'Charlie']]
3 -> [[17, 'David']]
4 -> []
5 -> []
6 -> [[20, 'Bob']]

Hash Table (Linear Probing):
0 -> None
1 -> None
2 -> (30, 'Green')
3 -> (10, 'Red')
4 -> (17, 'Yellow')
5 -> None
6 -> (20, 'Blue')
Search key 17: Yellow

Hash Table (Linear Probing):
0 -> None
1 -> None
2 -> (30, 'Green')
3 -> (10, 'Red')
4 -> (17, 'Yellow')
5 -> None
6 -> None
```