CSCE 3110 Assignment 5
Due: 11/24, 11:59PM

Question : Implement a Linear Probing Hash Table

Write a program with C++ to implement a hash table using linear probing for collision handling. Your hash table should:

1) Support the following operations:

   insert(key, value): Insert a key-value pair into the hash table. If the key already exists, update its value.

   delete(key): Remove the key-value pair for a given key from the hash table.

   get(key): Retrieve the value associated with a given key.

   resize () (for dynamic resizing)

   display (): display the hash table

2) Handle collisions using linear probing.
3) Automatically resize the hash table when the load factor exceeds 0.7.
4) Include a display() method to print the contents of the hash table.

Submission:

Code Documentation

A brief document or README file (in plain text, Word, or PDF format) explaining:

- **The design of the hash table.**
- **How linear probing was implemented.**
- **How resizing and load factor were handled.**

Screenshots of the program's code and output for the following test cases

Test Case:

```
Initialize hash table with capacity 5.

Operations:
1. Insert: ("John", 10)
2. Insert: ("Jane", 20)
3. Insert: ("Jack", 30)
4. Insert: ("Jill", 40)
5. Insert: ("Jerry", 50)  # Causes resizing since load factor exceeds 0.7
6. Insert: ("Jenny", 60)
7. Get: ("Jane")
8. Get: ("Jenny")
9. Delete: ("Jack")
10. Insert: ("Jake", 70)
11. Insert: ("Jared", 80)
12. Display: Current Hash Table
```

Input/Output Example

Input:

Initialize hash table with capacity 5.

Operations:

1. Insert: ("Alice", 25)

2. Insert: ("Bob", 30)

3. Insert: ("Charlie", 20)

4. Insert: ("David", 35)

5. Insert: ("Eve", 50) # Causes resizing since load factor exceeds 0.7

6. Get: ("Charlie")

7. Delete: ("Bob")

8. Insert: ("Bob", 40)

9. Display: Current Hash Table

Output:

```
Initial Hash Table (Capacity: 5):

Index 0: None

Index 1: None

Index 2: None

Index 3: None

Index 4: None


After Inserting ("Alice", 25):

Index 0: Alice -> 25

Index 1: None

Index 2: None

Index 3: None

Index 4: None


After Inserting ("Bob", 30):

Index 0: Alice -> 25

Index 1: Bob -> 30

Index 2: None

Index 3: None

Index 4: None


After Inserting ("Charlie", 20):

Index 0: Alice -> 25

Index 1: Bob -> 30

Index 2: Charlie -> 20
Index 3: None

Index 4: None


After Inserting ("David", 35):

Index 0: Alice -> 25

Index 1: Bob -> 30

Index 2: Charlie -> 20

Index 3: David -> 35

Index 4: None


After Inserting ("Eve", 50): (Resized to Capacity: 10)

Index 0: Alice -> 25

Index 1: None

Index 2: Charlie -> 20

Index 3: None

Index 4: None

Index 5: Bob -> 30

Index 6: None

Index 7: None

Index 8: David -> 35

Index 9: Eve -> 50


Get ("Charlie"):

Value: 20


After Deleting ("Bob"):
```

```
Index 0: Alice -> 25
Index 1: None
Index 2: Charlie -> 20
Index 3: None
Index 4: None
Index 5: DELETED
Index 6: None
Index 7: None
Index 8: David -> 35
Index 9: Eve -> 50

After Re-Inserting ("Bob", 40):
Index 0: Alice -> 25
Index 1: None
Index 2: Charlie -> 20
Index 3: None
Index 4: None
Index 5: Bob -> 40
Index 6: None
Index 7: None
Index 8: David -> 35
Index 9: Eve -> 50

Final Hash Table:
Index 0: Alice -> 25
Index 1: None
Index 2: Charlie -> 20

Index 3: None

Index 4: None

Index 5: Bob -> 40

Index 6: None

Index 7: None

Index 8: David -> 35

Index 9: Eve -> 50
```