Ex : 10

Title: Design and develop a Program in C that uses Hash function H: K ->L as H(K)=K mod m and implement hashing technique to map a given key K to the address space L. Resolvethe collision (if any).

Problem Description: Given a set K of Keys (4-digit) which uniquely determines the records in file F. Assume that file F ismaintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers.

Method: Ensure that the program generates the 4-digit key randomly and generates the L using hash function, demonstratesif there is a collision and its resolution by using linear probing.

Theory Reference: Module 5

Explanation:

1. **Linear Probing**: Collisions are resolved by checking the next available index (index1 = (index1 + 1) % m;), which continues until an empty slot (-1) is found.
2. **Memory Allocation**: The hash table (a) is dynamically allocated using malloc.
3. **Handling Table Full Condition**: If the hash table is full, it stops inserting further keys and reports the issue.

Algorithm:

Step 1: Initialize the Hash Table

1. **Create an array** `a` of size `m` (in this case, `m = 20`), and initialize all elements to `-1` to represent empty slots.
2. Set `count = 0` to keep track of the number of elements inserted into the hash table.

Step 2: Input the Number of Keys and Validate

1. Ask the user for the number of keys to insert into the hash table.
2. **Validate the input**:

- If the number of keys `n` is greater than `m`, print an error message and terminate the program (since the hash table cannot accommodate more keys than its size).

Step 3: Input the Keys

1. Ask the user to input `n` keys (each key is a 4-digit integer).
2. Store the keys in an array `key[20]`.

Step 4: Insert Keys into the Hash Table Using Linear Probing

For each key in the array `key[]`:

1. **Compute the index** where the key should be inserted:
   - Use the hash function: `index1 = key % m` (this gives an index in the range `0` to `m-1`).
2. **Linear Probing** to resolve collisions:
   - Check if the slot at index `index1` is occupied (`a[index1] != -1`):
     - If occupied, move to the next slot: `index1 = (index1 + 1) % m`.
     - Repeat this process until an empty slot is found (i.e., `a[index1] == -1`).
3. **Insert the key**:
   - Once an empty slot is found, insert the key at that index: `a[index1] = key`.
   - Increment the `count` to indicate that a key has been successfully inserted.
4. **Check if the hash table is full**:
   - If `count == m`, print a message indicating the table is full and stop inserting further keys.

Step 5: Display the Hash Table

1. Traverse the array `a[]` and print the keys stored in the hash table. For each index:
   - If the slot is empty (`a[i] == -1`), print "Empty".
   - Otherwise, print the key stored at that index.