OIC Computer Science Year 2

Hash Table Assignment

Introduction
A hash table is an abstract data type that is used to map a key to a value.
See here for more detail: https://en.wikipedia.org/wiki/Hash_table
See here for much more detail:
https://www.cs.cornell.edu/courses/cs3110/2014fa/lectures/13/lec13.html

One very common use of a hash table is to store records in an array or list so that records can be very quickly retrieved on demand – for example looking up a website user's details when they log in.

The aim of this exercise is to implement a small hash table so that you can understand how it works.

Task

Assume we want to store customer records in a 1D array.

Each customer has a unique ID an integer in the range 10001 to 99999.

The customer record consists of just two items, the customerID and the customer's name. The customerID field will be used as the key field.

We need to store records for the following customer IDs:

45876, 32390, 95312, 64636, 23467

To keep things simple a hash table of just 10 records will be used for this exercise.

1. Create a class to represent a customer record.

The class should have an attribute for each field in the record.
As well as the constructor, the class should define a method to display all the fields in the record.

2. Create a class to represent the hash table.

The class should include the storage for the table and the following methods:

Method	Description
__init()__	The constructor. Creates the store for the hash table and performs any other initialisation needed.
insertRec()	Inserts a record into the hash table.
findRec	Searches the hash table to find the specified record.
hash()	A hashing function that converts a key value into a location value in the store.
getCollisions()	A function that returns the number of collisions that have occurred as

	records are added to the store.

3. Write a function to test the hash table

The function should:

- create five records that contain the customerIDs mentioned above.
- Create a hash table
- Insert all of the records above into the hash table
- Display the contents of the hash table to check that the records have been inserted correctly and that the collision management technique works correctly.
- Search the hash table for each of the test records, confirming that they are found.
- Search for a record that is not in the hash table.

When inserting records into the table, collisions will occur because the data has been deliberately chosen to create them. These should be resolved using the open addressing (aka closed hashing) technique with linear probing. This technique uses sequential searching of the store beyond the point of collision until an empty slot is detected.

(see next page for pseudocode)

Pseudocode

```
FUNCTION hash(key)  RETURNS INTEGER
  RETURN key MOD tableSize
ENDFUNCTION


PROCEDURE insert(newRecord)
  index = hash(newRecord.key)
  # deal with collisions using open addressing & linear probing
  WHILE hashTable[index] NOT empty
    index = index +1
    If index > tableSize THEN
      index = 0
    ENDIF
  ENDWHILE
  # index now indicates an empty slot
  hashTable[index] = newRecord
ENDPROCEDURE


FUNCTION findRecord(searchKey) RETURNS record
  index = hash(searchKey)

  WHILE hashTable[index].key <>  searchKey AND hashTable[index] NOT empty
    index = index +1
    If index > tableSize THEN
      index = 0   # wraparound
    ENDIF
  ENDWHILE

  IF hashTable[index] NOT empty
    RETURN hashTable[index]   # record found
  ELSE
    RETURN None    # record not found
  ENDIF
ENDFUNCTION
```