MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG315 Algorithms

Assignment 2: Transaction Data Management with a Hash Table

This assignment aims to help you practice **hash tables**, especially collision resolution with **open addressing**. Your main task in this assignment is to develop a hash table to store some of the customers you read in the first assignment using **C programming language**.

Overview

Let's say you want to keep the customers you read in the first assignment stored in a database, but you do not want them to be stored so blatantly in a normal table, especially when considering the performance advantages of hash tables when your system would hypothetically need to keep track of thousands of customers.

In this assignment, you will be developing a hash table with specific hashing techniques. The system shall be able to perform the following functionalities:

1. Read customers from files and put them in a hash table
2. Search for a customer by taking their name
3. Print table

Preset

You will not be required to write the code for reading the customers from the transactions file in this assignment. Instead, the code for **countCustomers** and **readTransactions** from the previous assignment will be given to you in the preset template, and you will be required to implement new functions concerning with hashing specifically. Below is an explanation of both functions in case you missed in the first assignment:

- **countCustomers:** This function takes the file pointer of the transactions file as a parameter and reads the content, and as it is doing so, it counts the number of unique customers found in the dataset and returns it. This number is then used in the main function to dynamically allocate the memory for the customers array.
- **readTransactions:** This function reads the transactions file again but this time, it reads and processes customer information as it populates the customers array. Every time it reads a data line, it checks if the customer in that line has already been added to the array, if yes, then it updates their info in the array, otherwise it adds them in a different spot.

Input

The program should take input commands from the user and data in a text file called "transactions.txt" containing transaction data in the following format:

```
UserName;TransactionId;TransactionTime;ItemDescription;NumberOfItemsPurchas
ed;CostPerItem;Country
Alice;6355745;02/02/2019;LONDON BUS COFFEE MUG;6;11.73;United Kingdom
Bob;6283376;12/26/2018;SET 12 COLOUR PENCILS DOLLY GIRL;3;3.52;France
Charlie;6385599;02/15/2019;PACK OF 12 SUKI TISSUES;72;0.9;Germany
...
```

Please note that the content of the data file can be changed, but the name of the file will not be changed, and the structure should be in the format as shown above.

Requirements

As the program reads the data, it maintains a customer structure for each unique customer and updates the total number of transactions, total number of items purchased, and total amount of money spent for each user. These customer structures are then stored in a dynamically allocated array.

Here is where your task in this assignment comes in; you are asked to create a hash table and ask the user for which open addressing method to use. Afterwards, you will iterate over the array of customers, adding each customer to the hash table one at a time based on the user's choice of open addressing and through the given hash functions. Should the load factor exceed 0.5, you will have to do rehashing.

Finally, the user can also search for customers in the hash table, in which you will have to use hashing in order to locate the customer; linear search is not allowed.

- **Hash table Implementation:** The application shall create a hash table where the user decides the collision resolution technique:
  - If the user enters 1, linear probing will be used where f(i) = i
  - If the user enters 2, quadratic probing will be used where $f(i) = i^2$
  - If the user enters 3, double hashing will be used where $f(i) = i * hash_2(x)$

- The initial size of the hash table is set to **11**. If the load factor (total number of customers in the hashtable / hashtable size) exceeds 0.5, rehashing is initiated with the following steps:
  1. Compute the size of the new hash table by doubling the size of the old hash table and rounding it to the next prime number.
  2. Dynamically allocate a new hash table and relocate customers properly.
  3. Destroy the old hash table.

- **Hash Functions:**
  key = |ASCII (last_character(customerName)) - ASCII (first_character(customerName))|
  hash(**key**) = **key** mod **hashtableSize**
  $hash_2$(**key**) = 7 - (**key** mod 7)

  Note: when calculating the key, make sure the first and last letters of the customer's name are in the same case; e.g. when calculating the key for Bob, the key has to be 0.

In this assignment, you are given a C file called "template2.c" attached. This will be the template code which you need to strictly follow. Inside the "template2.c" file, all function prototypes are declared, the main function is already written along with a few others. You will basically need to fill in the blanks and write the code for the following functions:

- **createHashTable**: This function simply creates an array of size 11 dynamically to represent the initial hash table and returns it. **Hint**: You could make some initializations here to help you in later parts of the code such as initialising the empty spots in the hashtable to name = "unassigned", transactions = items_purchased = amount_paid = 0.
- **addCustomer**: This function takes as input the hash table, the customer to be added, the hash table's capacity (the current number of customers in the hash table), its size, and the hashing criteria (linear probing, quadratic probing, and double hashing) while returning the new hash table at the end after adding the customer. Note that whenever the load factor exceeds 0.5, you will need to call **rehash** inside this function.

- **rehash**: This function takes as input the hash table and its size. It finds the new size (double the old size and round it to the new prime number), creates a new table, and rehashes the customers in the old table to the new table before destroying the former and returning the latter.
- **printTable**: This function takes the hash table and its size and displays it.
- **searchTable**: This function takes the hash table, its size, a customer's name, and the hashing criteria (linear probing, quadratic probing, and double hashing) and searches for the specified customer in the table, displaying their information when found, and an error message when not.

Process Model:

1. Hash table creation.
2. User chooses linear probing, quadratic probing, and double hashing for collision resolution.
3. Read customer data: Process data and insert customers into the hash table, rehashing when necessary, and displaying the hash table after each insert.
4. User options:
   A. Search for a customer: Allow users to search for customer information based on various criteria, displaying details if found.
   B. Display hash table: Print the contents of the hash table.
   C. Exit the program.

Implementation Notes:

- Rehashing must involve recomputing positions for each element according to the new hash table size.
- Usage of linear search when searching a customer will result in an instant 0 (zero) for the search part, please ensure you use hashing methods to find the target customer.
- When printing the hash table, you should display the open empty spots.
- The template code assumes the data file you are going to read is called "transactions.txt" and is stored directly in the code's immediate directory. Note that if you are using an IDE like Clion, you will need to store the file in the "cmake-build-debug" folder.
- You are not allowed to change any code in the template, and you should abide by the prototypes and structures given. The places where you need to find write your code are marked with the comment: "WRITE YOUR CODE HERE".
- There are data columns in the file that you do not need for your program such as the transaction time and country, so you do not need to read those data columns.
- You need to ensure usage of comments to clarify your codes.

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow **modular programming** which means that you are expected to divide your program into a set of meaningful functions. Specifically, you are expected to implement some helper functions.

Output

You can find a sample output in the text file attached to the assignment called sample_output.txt.

Submission

You will submit only a single .c file to ODTUCLASS. The name of the .c file should be your 7-digit student id number.

Grading Scheme

The assignment will be graded based on the following scheme:

Grading Item	Mark (out of 100)
Hash table creation	5
addCustomer	35
rehash	30
printTable	10
searchTable	20

The assignment will be graded as follows:

- **addCustomer** and **rehash** will be tested with a different transaction data file with different hashing criteria. If a particular hashing criterion does not create an expected output, then **you will not receive any mark from that option.**
- If other functions do not work as expected, then **you will not receive any mark from those functions.**

You need to ensure your code runs and is testable. **If you submit a code which does not compile, you will automatically get zero.** Please note that code quality, modularity, efficiency, maintainability, and appropriate comments will be part of the grading. Therefore, providing an expected output does not guarantee a full mark.

Professionalism and Ethics

You are expected to complete the assignments on your own. Sharing your work with others, uploading the assignment to online websites to seek solutions, and/or presenting someone else's work as your own work will be considered as cheating.

Please note that we will use a tool to compute the similarity between your submissions, and there are also tools to detect if the code you submitted is generated by AI tools.