MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG315 Algorithms

Assignment 1: Subscription Data Management with a Hash Table

This assignment aims to help you practice **hash tables**, especially collision resolution with **open addressing**. In this assignment, you will develop an application that manages subscriptions for different users. The goal is to create a system that efficiently stores, retrieves, and manages various subscriptions (such as streaming services, gym memberships, etc.) using hash tables. You will implement collision resolution strategies to ensure optimal performance as the number of subscriptions grows. You will implement the assignment in the **C programming language**.

Overview

Using a hash table, you will implement an efficient system for managing customer and subscription data. Storing information in a simple table may not scale well, especially when handling many customers. Using a hash table can enhance performance, allowing for fast lookups and data management.

Your task is to design a system that:

- Reads customer and subscription data from input files.
- Creates a hash table of customers, storing their details, tracking the number of subscriptions, and updating the total amount spent by each customer. The user will select the collision resolution technique (linear probing, quadratic probing, or double hashing).
- Handles automatic rehashing when the load factor exceeds 0.5, expanding the hash table size to maintain efficiency.
- Supports customer searches using hash functions, avoiding the linear search for lookups.

By the end of this assignment, you will have developed a scalable and efficient data storage solution leveraging hash tables, capable of handling customer information with high performance.

Input

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

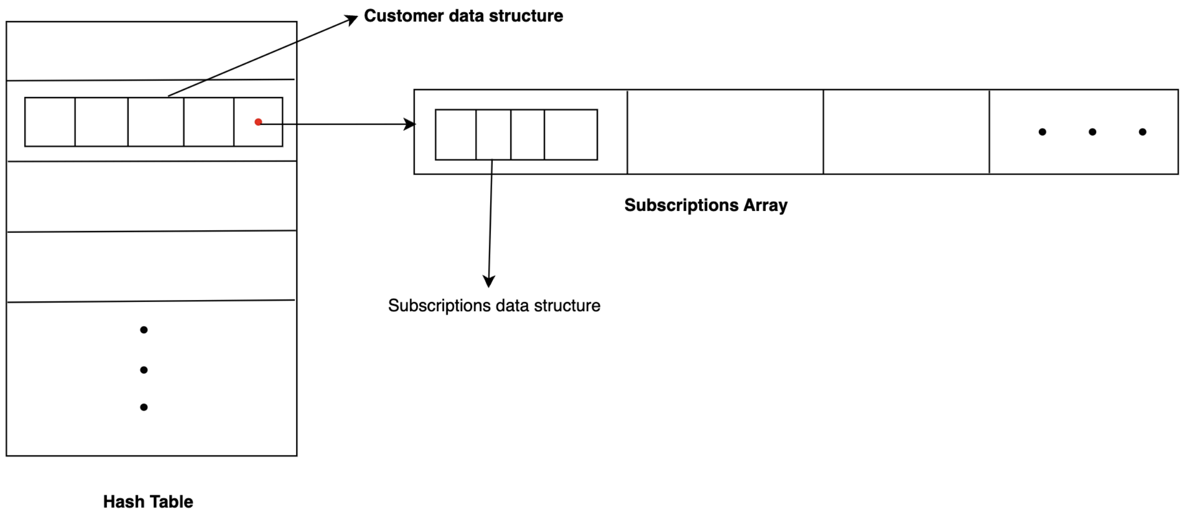
```
UserName;SubscriptionID;ServiceName;ServiceCharge;DevicesRegistered;StartDate;EndDate
;Status;Country
Alice;333;Netflix;15.99;3;2023-01-01;2023-12-31;Active;United Kingdom
Bob;234;Spotify;9.99;2;2023-05-01;2024-04-30;Active;Germany
Charlie;127;Amazon Prime;12.99;4;2022-03-15;2023-03-14;Expired;France
Alice;245;Spotify;9.99;2023-02-02;2023-12-20;Active;United Kingdom
...
```

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 shown above.

Requirements

The program reads customer and subscription data from the input file and constructs a **Customer** structure for each unique customer. Each customer structure includes the total number of subscriptions, the total amount spent, and an array of **Subscription** structures detailing each service the customer has subscribed to. These customer structures are then added to the hash table based on the chosen open-addressing method for collision resolution.

Here's a visualization of the hash table (that will contain Customer structures) and the Subscription array for each Customer. These data structures have already been provided to you in the template.c file.



Here is where your task in this assignment comes in; you are asked to create a hash table and ask the user which open addressing method to use. Afterwards, you will iterate over the array of customers, adding each unique 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.

In short, your task is to:

- **Read the subscription data** and store it for each unique customer
- **Create a hash table** using dynamic memory allocation.
- **Insert each unique customer** into the hash table along with their subscription details, using the user-selected collision resolution technique.
- **Rehash the table** if the load factor exceeds 0.5 to ensure efficient handling of future insertions.
- **Search for customers** in the hash table using hashing, avoiding linear search.

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:

For the key calculation, you are asked to use **XOR hashing. XOR hashing** is a technique where you XOR the ASCII values of all characters in the string.

Let key = 0 initially
For each character c_i in the input string (customer name):
key = key $\oplus$ ASCII(c_i)

hash(**key**) = **key** mod **hashtableSize**
$hash_2$(**key**) = 7 - (**key** mod 7)

In this assignment, you are given a C file called "template.c" attached. **This will be the template code that you need to strictly follow**. Inside the "template.c" file, all function prototypes are declared, and functions such as **main** and **printCustomers** are already written. You will need to fill in the blanks and write the code for the following functions:

- **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 customer array's memory.

- **readSubscriptions:** This function takes the file pointer of the transactions file, the array of Customers, and the number of customers. The function reads and processes customer information as it populates the customer's 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.

- **createHashTable**: This function simply dynamically creates an array of size 11 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 initializing the empty spots in the hashtable to name = "unassigned", etc.

- **addCustomer**: This function takes as input the hash table, the customer to be added, the hash table's 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, the hashing criteria, 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. Note that it displays all of the hash table (empty places in it as well. Check sample output)

- **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 subscription information when found**, and an error message when not (for the information to be displayed check sample output).

Implementation Notes:

- Rehashing must involve recomputing positions for each element according to the new hash table size.
- The use of linear search when searching for 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 "subscriptions.txt" and is stored directly in the code's immediate directory. Note that if you use 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 and write your code are marked with the comment: "WRITE YOUR CODE HERE".
- You need to ensure the 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 the sample outputs in the text files attached to the assignment.

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)**
countCustomers	15
readSubscriptions	20
Hash table creation	10
addCustomer	20
rehash	10
printTable	10
searchTable	15

The assignment will be graded as follows: Each function will be tested with different inputs and if your function does not work as expected then **you will not receive any marks from that function.**

You need to ensure your code runs and is testable. **If you submit a code that 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

Students are expected to complete the assignments/exams on their own. Sharing your work with others, uploading the assignment/exam questions to the online websites for seeking solutions, and/or presenting someone else's work/solution as your work will be considered cheating. In addition, using generative AI tools to generate solutions and/or presenting generative AI solutions as one's original work will also be considered 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.