Banking System

Team Member:

1. Angeline Karen (2440035601)
2. Vania Natalie (2440032423)
3. Michael Christopher (2440047362)

I. Problem Description

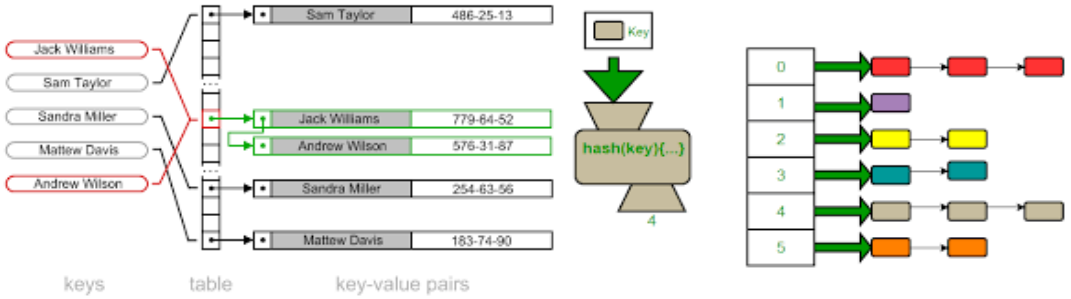
A structured program is essential in writing codes that are used and edited by many programmers. It is difficult to collaborate if there isn't any structure in the code, especially since each programmer has their own style of coding. In terms of a banking system, a common concern is the safety of the customer's information. In a banking system, we need to give up our personal and account information. The banking system has to guarantee the safety of the sensitive information of the customer, especially from the admin, who has the privilege to access some general information from the customer's account. The banking system also has to have an organized way of storing the data, where the data can be easily searched. For our final project, we are going to implement a structured program for a simple banking system. We will use object-oriented programming to implement the codes.

II. Alternative Solutions

- Connecting to a database. Using a database such as MySQL to the program. In a huge project where we need to store a lot of data, using a database is a good choice. However, since we are only implementing a simple banking system, using a database takes up more storage, and to access or search the data takes more time as the database needs to search each data one by one and compare it. It makes the processing slower than a regular hash table.
- Hash table using linear probing for handling collision. Linear probing is a very simple method for handling collision, where we store the collided data into the next index (until an empty index is found) if the hashed index is taken / collided. Although the implementation is fairly simple, this is not the optimal solution for handling collisions. It can lead to clustering and lots of collisions in the hash table.

III. Theoretical Analysis of The Data Structures

For our banking system project, we chose to use the hash table data structure to save the customer data. A hash table is a data structure that contains an array of abstract data types, a structure that maps a specific unique key to values. A hash table uses a hash function to compute an index into an array to find the desired data. The key is hashed and the result indicates which index the corresponding value is stored in the hash table. Although a hash table has a pair of one unique key and values, most hash tables have an imperfect hash function, which might cause collisions. This happens when the hash function generates the same index for more than one key. There are various ways to handle these kinds of collisions. One way is by using linear probing as mentioned above. The other way, which is also the method we implement in our project, is using chaining with a linked list for handling collisions. When a collision happens, the data is linked after the last element of the hashed index, instead of saving the data in the next empty index. The index then may contain unlimited data



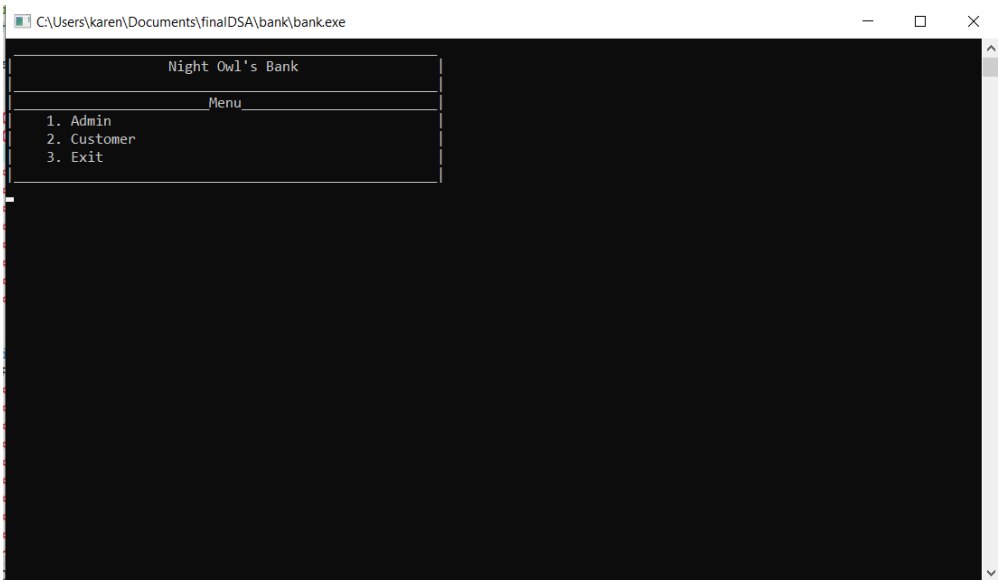
because of the linked list, and the collision will be handled. On average, hash tables turn out to be more efficient than search trees or any other table lookup structure. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and sets.

Next, we also implement a simple encryption and decryption in our banking system for storing the password and pin inside the hashtable. Encryption is the process of converting a normal message (plaintext) into a meaningless message (ciphertext) and decryption is the process to revert the ciphertext back to plaintext. It is to make our system safer since the customer's password and pin are encrypted before being stored inside the hashtable. This makes it harder for people who do not have access to the hash table (like hackers) to steal the customer's data.

IV. How to Execute (Program Manual) and Results of Execution

Initially, the program shows this "Night Owl's Menu" when executed. The menu is consisted of:

1. Admin menu: to access all the menu options for admin users
2. Customer menu: to access all the menu options for customers
3. Exit: to terminate the program



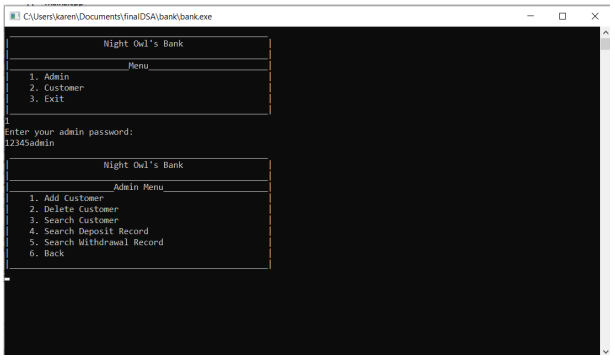
a) Admin Menu

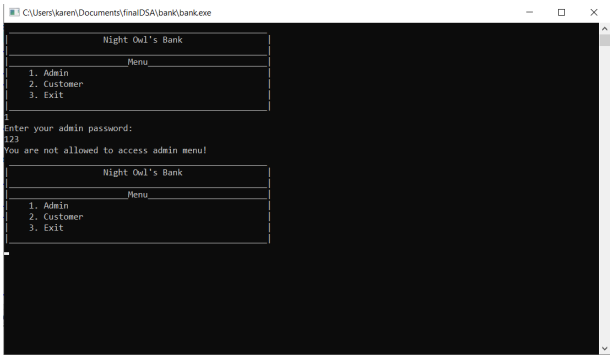
Here, in the Admin menu, there are six menu options from which the admin can choose.

- Add Customer: to add new customer
- Delete Customer: to delete a certain customer by their account key
- Search Customer: to search and display certain customer's information by their account key

- Search Deposit Record: to search certain customer's deposit record by their account key
- Search Withdrawal Record: to search certain customer's withdrawal record by their account key
- Back: to go back to the main menu

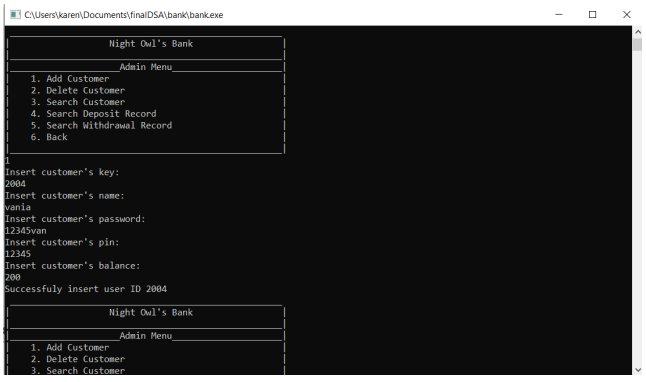
However, this admin menu can only be accessed through the admin's password.

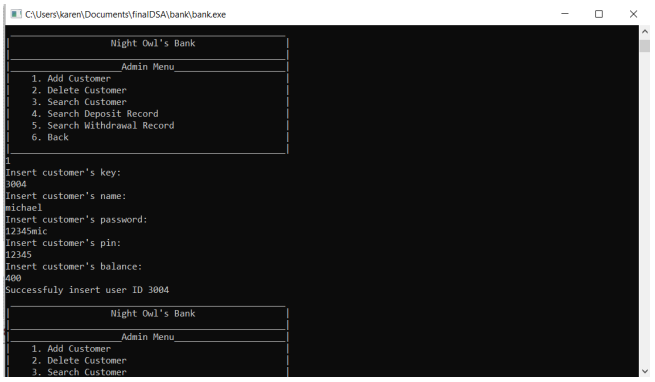


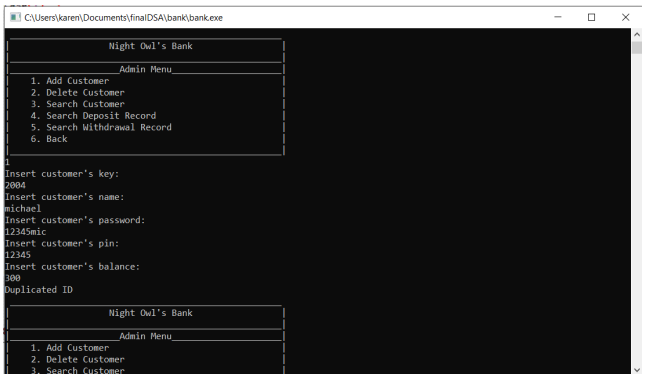


1. Add Customer

The Add Customer option is used to add the new customer into our hashtable. If the customer's key is placed in the same index, it will be linked to the last element of the index to avoid a collision. Additionally, if the admin input a duplicate user's key, it will generate an error message.

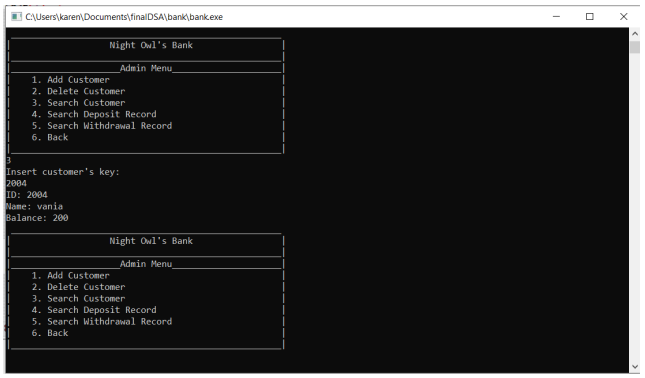


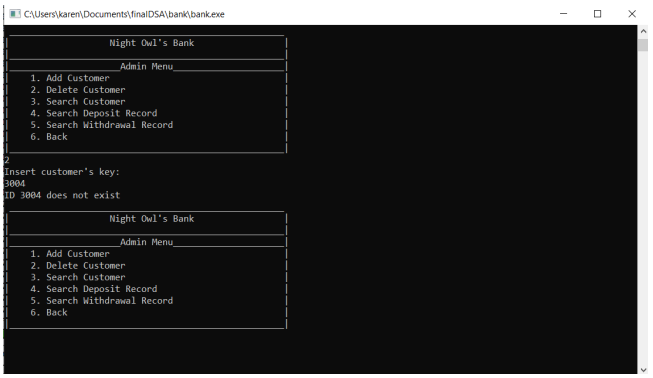




2. Delete Customer

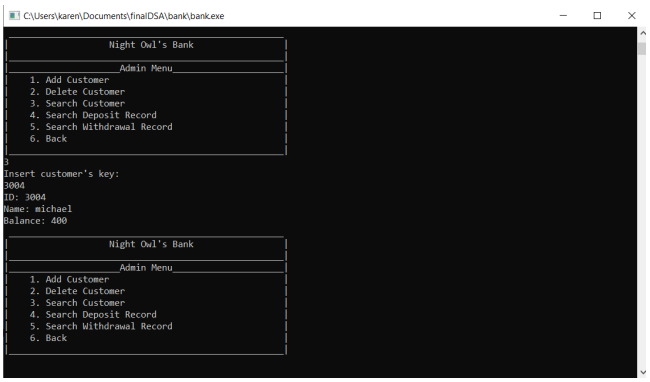
The delete customer option is used to delete a customer by a certain customer's key. Suppose we have created a user with the user's key of 3004. If the admin tries to delete a non-existent user, there will be an error message showing that the user with a certain customer's key doesn't exist. Else, the customer with key 3004 will be deleted.

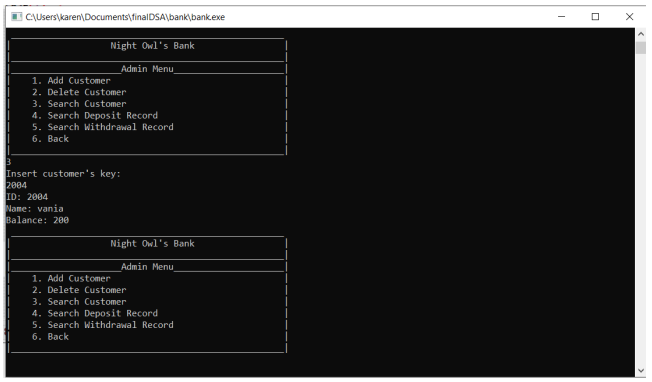


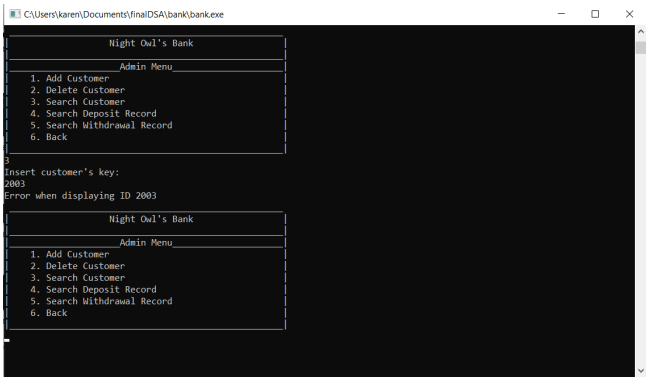


3. Search Customer

The search customer option is used to search a certain customer by their account key and display their account information. If the admin tries to search for a non-existent user, there will be an error message.

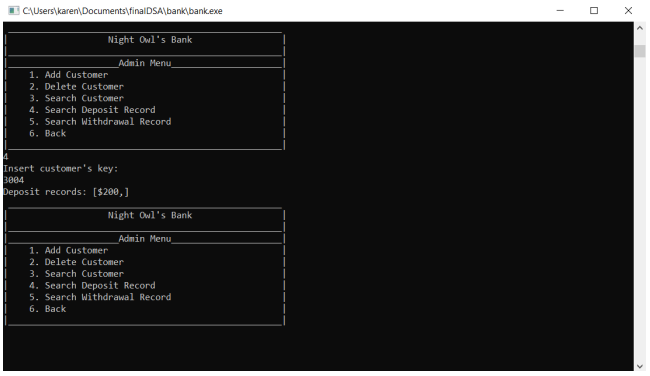






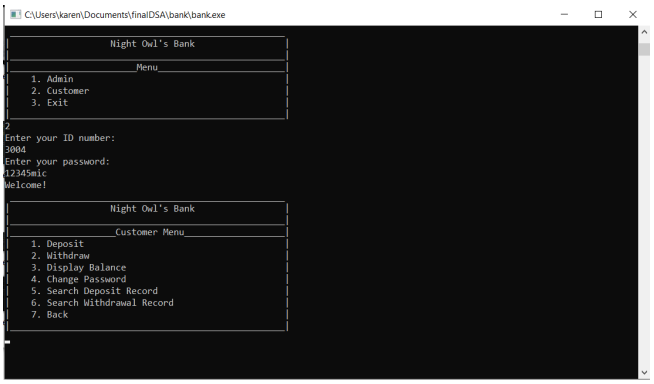
4. Search Deposit Record

The search deposit record option is used to show a certain customer deposit record. The admin can find it by inputting a certain customer's key they'd like to find.



5. Search Withdrawal Record

The search withdrawal record option is used to show a certain customer withdrawal record. The admin can find it by inputting a certain customer's key they'd like to find.



6. Back

The exit option is used to exit out from the admin's menu and go back to the main menu.

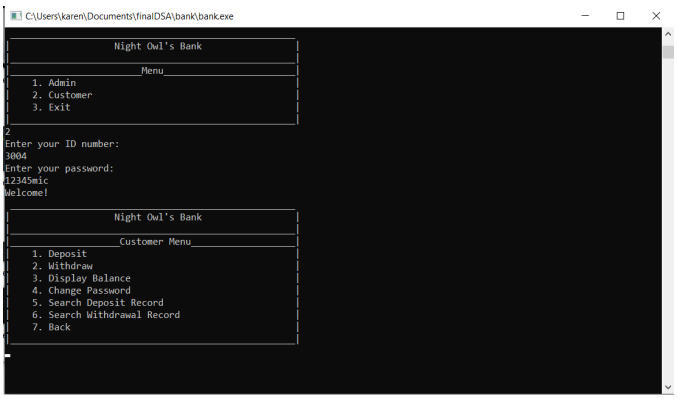
b) Customer Menu

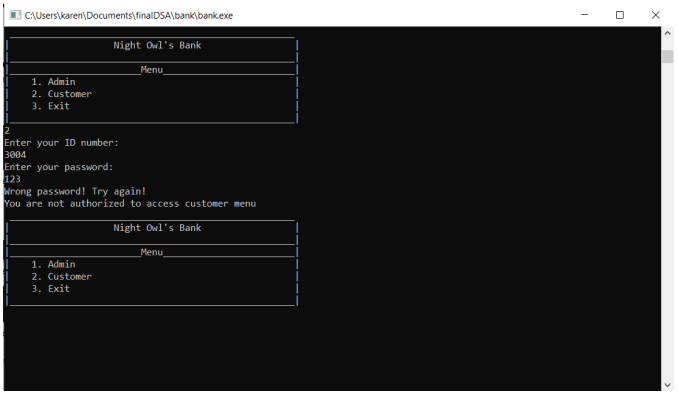
Here, in the Customer menu, there are six menu options from which the admin can choose.

- Deposit: to deposit
- Withdraw: to withdraw

- Display Balance: to display the current logged-in customer's balance
- Change Password: To change the current logged-in customer's password
- Search Deposit Record: to show the current logged-in customer's deposit record
- Search Withdrawal Record: to show the current logged-in customer's withdrawal record
- Back: to go back to the main menu.

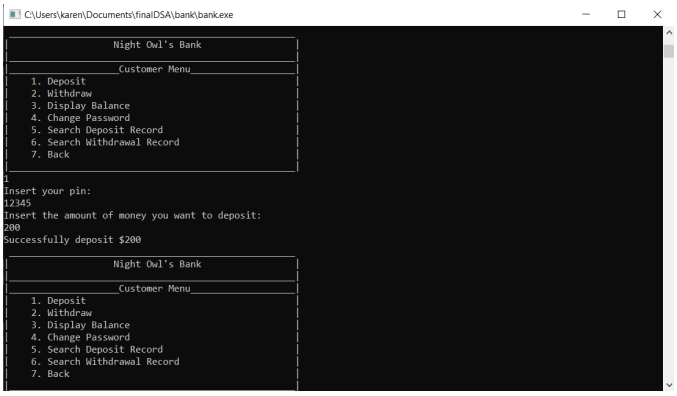
However, this customer menu can only be accessed through the customer's key and password.





1. Deposit

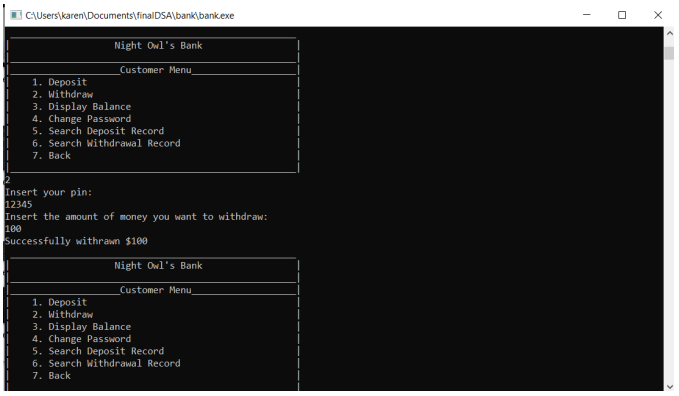
The deposit option is used to allow the currently logged-in customer to deposit a certain amount of money.

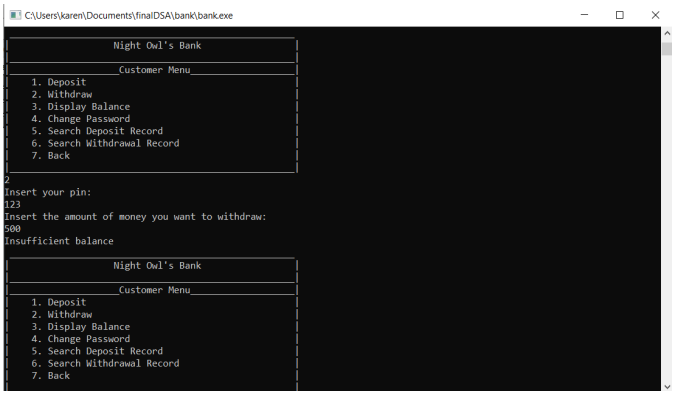


2. Withdraw

The withdraw option is used to allow the currently logged-in customer to withdraw a certain amount of money. If the customer tries to withdraw the

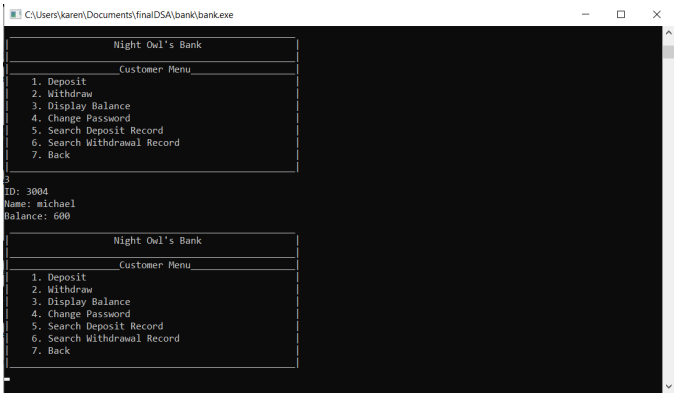
amount of money larger than their current balance, there will be an error message.





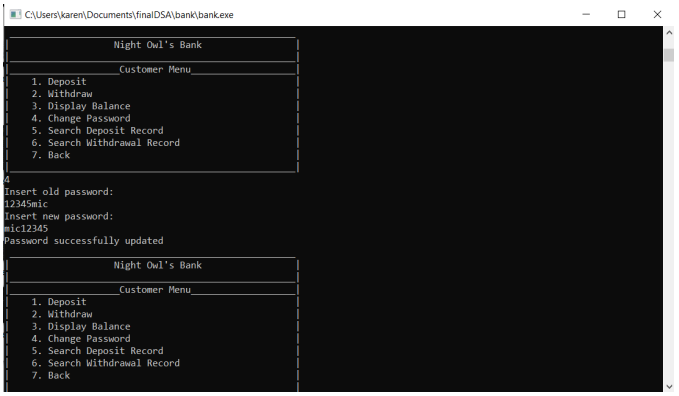
3. Display Balance

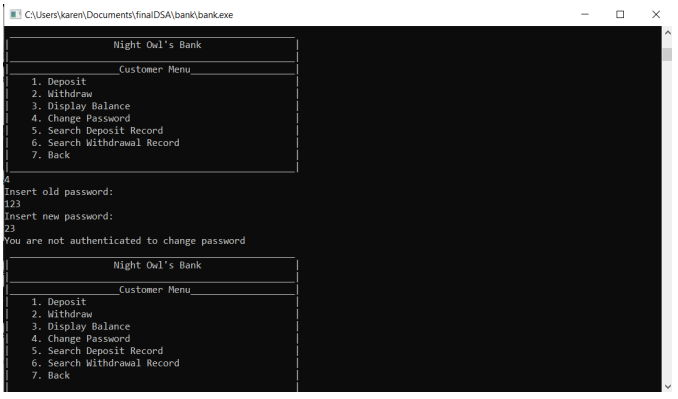
The display balance option is used to display the currently logged-in customer to display their current balance.



4. Change Password

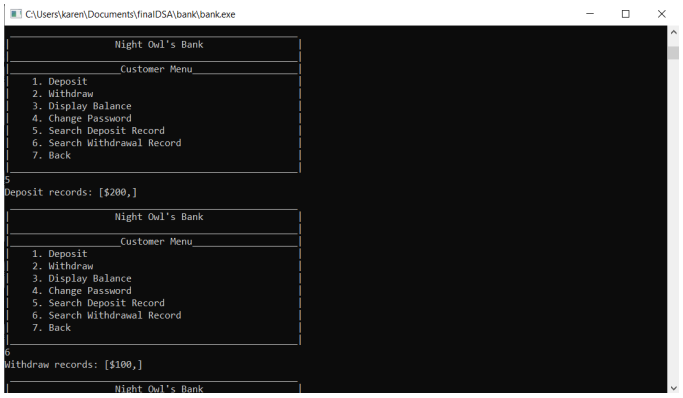
The change password option is used to change the currently logged-in customer to change their password. If the customer inputted the wrong old password, there will be an error message.





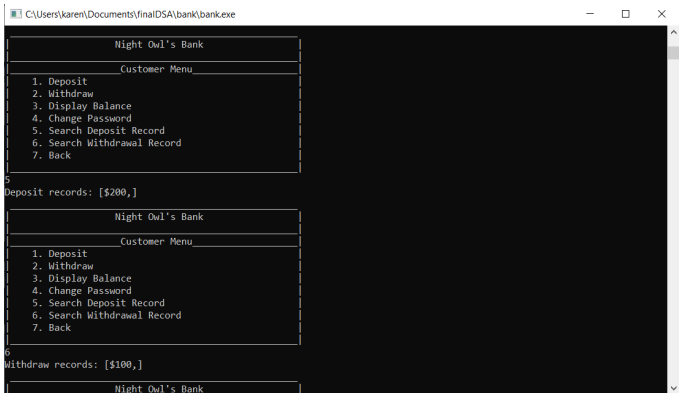
5. Search Deposit Record

The search deposit record option is used to show the currently logged-in customer their deposit record.



6. Search Withdrawal Record

The search withdrawal record option is used to show the currently logged-in customer their withdrawal record.



7. Back

The back option is used to exit out from the customer's menu and go back to the main menu.

V. Runtime Measurements

When the data is saved at the first element of the index (no chaining):

No	Function	Elapsed Time (millisecond)
1.	Insert customer	1.5498
2.	Display data	2.1494

3.	Deposit	0.515
4.	Withdrawal	0.3983
5.	Search Deposit Records	0.5378
6.	Search Withdrawal Records	0.5101
7.	Delete customer	0.7385
8.	Updating password	0.2307

When data is saved at second element of the same index (with chaining):

No	**Function**	**Elapsed Time (millisecond)**
1.	Insert customer	2.1883
2.	Display data	2.294
3.	Deposit	0.4755
4.	Withdrawal	0.2138
5.	Search Deposit Records	19.0807
6.	Search Withdrawal Records	4.0633
7.	Delete customer	0.8006
8.	Updating password	12.3302

From both of the tables above, the elapsed time (millisecond) for both deposit and withdrawal functions doesn't have a significant difference.The elapsed time for the other functions has a difference, with the linked data taking a bit longer to execute.

The average elapsed time for executing the functions:

No	**Function**	**Average Elapsed Time (millisecond)**
1.	Insert customer	1.87

2.	Display data	2.22
3.	Deposit	0.50
4.	Withdrawal	0.31
5.	Search Deposit Records	9.81
6.	Search Withdrawal Records	2.54
7.	Delete customer	0.77
8.	Updating password	6.28

VI. Time and Space Complexity Search Function

	Time Complexity	Space Complexity	Explanation
Best Case (Big Ω)	O(1)	O(1)	Happens when there is only 1 digit in the key
Average Case (Big Θ)	O(n)	O(n)	Happens when there is more than 1 digit in the key. The number of digits is n. On average, the key will have more than 1 digit.
Worst Case (Big O)	O(n)	O(n)	Happens when there is more than 1 digit in the key. The number of digits is n. If the number of digits in n is extremely big

VII. Why We Chose Hash Tables with Chaining and the Hash Function

We chose a hash table with chaining (linked list) because our alternative solutions do not meet our criteria in making an efficient way of storing customer's data. Using a database will require an external application to connect to the program.

The searching in the database will also be done by comparing the key (customer id) with the id in the database one by one. This searching will get slow if there's a lot of data to compare with. A database is optimized for storing a large amount of data while a hash table is optimized for an efficient way of searching data; and the bonus is a hash table can store data too. The second alternative solution we proposed is using linear probing as a way to handle the collision. Although using linear probing for smaller amounts of data is more efficient compared to using chaining, we soon realized that the searching function is slow when there is a larger amount of data. Linear probing handles collisions by inserting the data to the next empty index; that means we also need to compare the data one by one when searching for a particular data. Using the chaining / linked list method for handling collisions provides a more efficient way of searching the specific data, as we just need to compare the data that is linked in one index after the key is hashed (by the hash function). For these reasons, we chose to implement a hash table with chaining, as we believe it's the most efficient way of searching and storing data for our banking system.

The hash function that we used is key % max_hash_size, which is 50. This is because it is the simplest hash function to get the index referenced in the hashtable. We considered using key % 10 for our hash function before, but we felt it was too simple and too easy to guess. This is why we chose key % max_hash_size. It is a simple enough hash function that if we were to change the maximum size of the hash table, it will have a different result.

VIII. Link to Video

The link to our video:
https://drive.google.com/file/d/12BLzkvc2bYShv3LU_O1YJyHS7ormTiUG/view?usp=sharing

IX. Link to Github

The link to our GitHub:
https://github.com/EuphosiouX/finalDSA