### 

### 

### 

### 

|  |  |
| --- | --- |
| **Abdelrahman Reda Fawzy** | **8716** |
| **Carl Sherif Kheir Eskander** | **8615** |
| **Michael Matta Abdrabou Nazeer Ibrahim** | **8957** |
| **Khaled Mohamed Bishr** | **8796** |
| **Daniel Wadie Motie Aziz** | **8663** |

### 

### **Introuction**

This C program serves as a simple banking system that allows users to perform various operations related to accounts, such as adding, modifying, withdrawing, depositing, transferring, and querying account details. The program uses a menu-driven interface to facilitate user interaction.

### **MAIN**

First, we print from the main\_menu() function that allows the user to choose what operation to perform on their account. If the user chooses to log in (number 1), it calls the login() function; if they choose to quit (number 2), it breaks the code. If an invalid choice is entered, it returns to the main menu.

### **LOGIN**

Input:

* Username
* Password

Mechanism:

This function takes the username and password, opens the file to check their validity. If valid, it calls the menu() function; otherwise, it prompts the user to enter a valid username and password.

Errors:

If the entered username or password is invalid, the function prompts the user to enter valid credentials.

### **MENU**

Input:

User-selected operation number.

Mechanism:

Prints the user menu, where each function corresponds to a number. The user enters the number of the function, and a switch-case structure calls the corresponding function. Functions include ADD, DELETE, MODIFY, SEARCH, ADVANCED SEARCH, WITHDRAW, DEPOSIT, TRANSFER, REPORT, PRINT, QUIT, and a default option to repeat the options.

### **LOAD**

Input:

* File name

Mechanism:

This function reads account information from the user-specified file. It dynamically allocates memory for the accounts based on the number of accounts and returns them from the function. If the file is not found, it prints "File does not exist."

Errors:

If the specified file is not found, the program prints "File does not exist."

### **QUERY**

Input:

* Account number for query

Mechanism:

Requests the user to input an account number for querying. Utilizes a linear search algorithm to find the account. If found, prints details; if not, prints "Account not found."

Errors:

If the entered account number is not found, the function prints "Account not found."

Pseudocode:

Create a pointer to an array of account structures called accounts.

Call a function named load to load account data from a file and store the number of accounts in numAccounts.

Prompt the user to enter the account number they want to query.

Read the account number input from the user.

Remove any newline character from the input.

Loop through each account in the accounts array:

Compare the entered account number with the account number of the current account using strcmp.

If a match is found, break out of the loop.

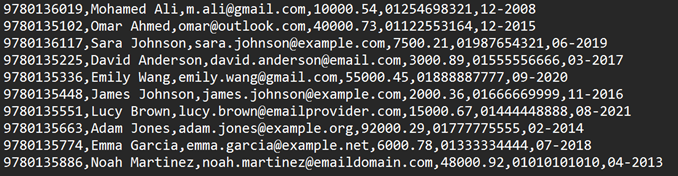
If the loop completes without finding a match, print a message indicating that the account is not found and recursively call QUERY() to allow the user to search again.

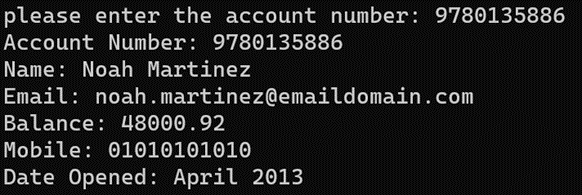
Print the details of the found account.

Free the allocated memory for accounts.

Call the menu() function to return to the main menu.

Sample Run:





### **ADVANCED SEARCH**

Input:

* Keyword for advanced search

Mechanism:

Requests the user to input a keyword for advanced search. Utilizes a linear search algorithm and prints details of found accounts. If none found, prints "Account not found."

Errors:

If no matching accounts are found, the function prints "Account not found."

Pseudocode:

Create a pointer to an array of account structures called accounts.

Call a function named load to load account data from a file and store the number of accounts in numAccounts.

Prompt the user to enter a keyword for searching.

Read the keyword input from the user.

Remove any newline character from the keyword.

Initialize i for loop counter

Initialize accountIndex array to store indices of found accounts

Initialize foundAccounts to count the number of found accounts

Convert the keyword to lowercase for case-insensitive search.

Loop through each account in the accounts array:

Convert the account name to lowercase.

Check if the lowercase keyword is found within the lowercase account name using a string searching function (strstr).

If found, store the index of the account in accountIndex and increment foundAccounts.

After the loop if the foundAccounts variable value still 0 then no accounts are found and the program prints a message indicating that no accounts match the keyword then recursively call ADVANCED\_SEARCH() to allow the user to search again.

Loop through the indices of found accounts in accountIndex:

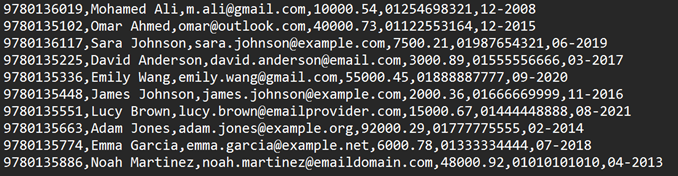
Retrieve the account details using the index.

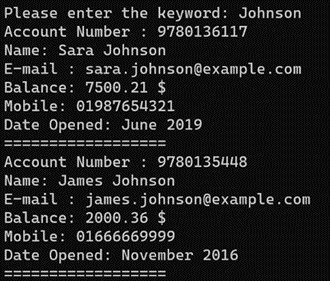
Print the account details, including account number, name, email, balance, mobile, and date opened.

Free the allocated memory for accounts.

Call the menu() function to return to the main menu.

Sample Run:





### **ADD**

Input:

* Account details (number, name, mobile number, email address, balance)

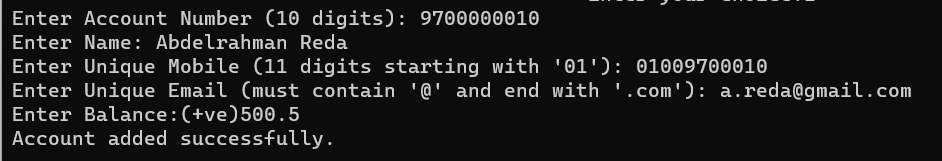
Mechanism:

Requests user input for new account details, validates the input, and appends details to the "accounts.txt" file. Prints a success message on completion.

Errors:

* If the file doesn't exist, prints an error message and exits the program.
* If the entered account details are invalid or duplicate, prints an error message and continues the input until validated.

Sample Run:



### **MODIFY**

Input:

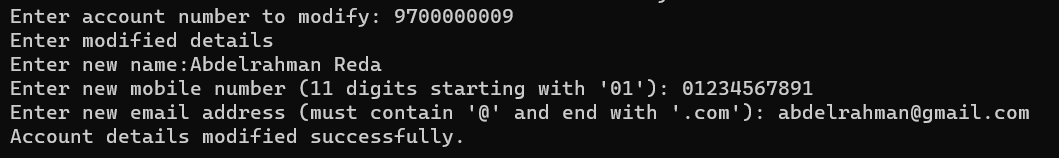
* Account number for modification

Mechanism:

Requests the user to input an account number for modification. Loads existing account details, compares the account number, and if found, allows the user to modify details. Saves modified details to "accounts.txt."

Errors:

If the entered account number is not found, the function prints "Account not found."

Sample Run****

### **DELETE**

Input:

* Account number to delete

Mechanism:

Asks the user to input an account number to delete. Handles account deletion conditions and saves modified account details to "accounts.txt."

Errors:

* If the account number does not exist, prints "Account not exist!" and continues the input loop.
* If the account balance is not zero, prints "Account balance must equal 0" and continues the input loop.

Sample Run:



### **WITHDRAW**

Input:

* Account number
* Amount of withdrawal process

Mechanism:

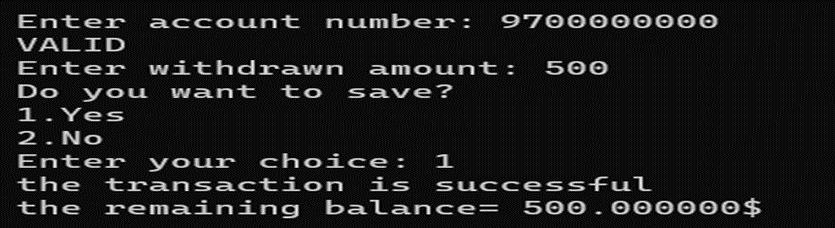
Call the function Account\* load (int \*num\_acc) to return all accounts and their number. Check the existence of the account by a for loop for 0 to Num Accounts. If the account is found, the program checks the withdrawal conditions. Asks the user if they want to save the process. If confirmed, decrease the account’s balance by the amount and call the function void save (Account \*acc2,int num\_acc) to save the new account details in the file “accounts.txt”.

After successfully doing the process, it prints a success message with the remaining balance. Create or append to a file with the account number’s name and print the process done. Closes the file after and frees the allocated memory and returns to the menu() function.

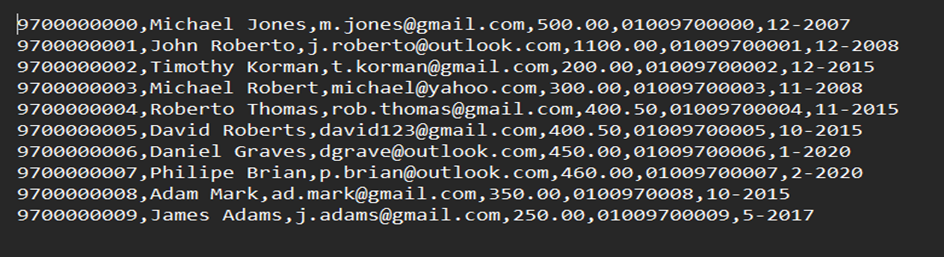
Errors:

* If the entered account number does not exist, prints an error message, and recursively calls the function again.
* If the entered amount is negative or more than the account balance or more than $10,000 prints an error message that the transaction is not successful. If the user enters a number not asked in the save question, it prints an error message and loops until the user enters it correctly.

Sample Run:



“accounts.txt” file after saving the process



The “9700000000.txt” file after saving the process



### **DEPOSIT**

Input:

* Account number
* Amount of deposit process

Mechanism:

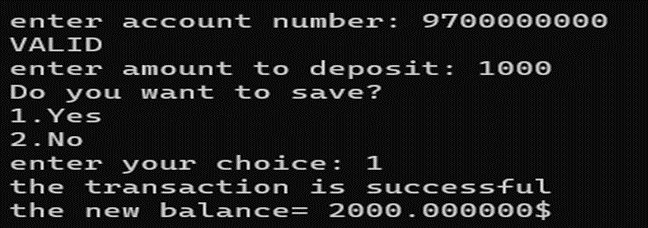
Call the function Account\* load (int \*num\_acc) to return all accounts and their number. Check the existence of the account by a for loop for 0 to numAccounts. If the account is found, the program checks the deposit conditions. Asks the user if they want to save the process. If confirmed, increase the account’s balance by the amount and call the function void save (Account \*acc2,int num\_acc) to save the new account details in the file “accounts.txt”.

After successfully doing the process, it prints a success message with the new balance. Create or append to a file with the account number’s name and print the process done. Closes the file after and frees the allocated memory and returns to the menu() function.

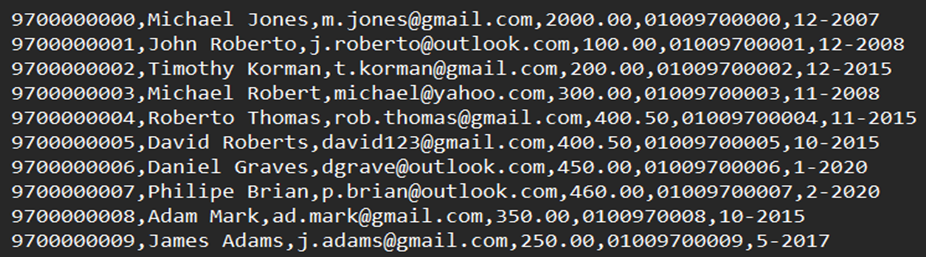
Errors:

* If the entered account number does not exist, prints an error message, and recursively calls the function again.
* If the entered amount is negative or more than $10,000 prints an error message that the transaction is not successful. If the user enters a number not asked in the save question, it prints an error message and loops until the user enters it correctly.

Sample Run:



The “accounts.txt” file after saving the process



The “9700000000.txt” file after saving the process



### **TRANSFER**

Input:

* Sender’s account number
* Receiver’s account number
* Amount of transfer process

Mechanism:

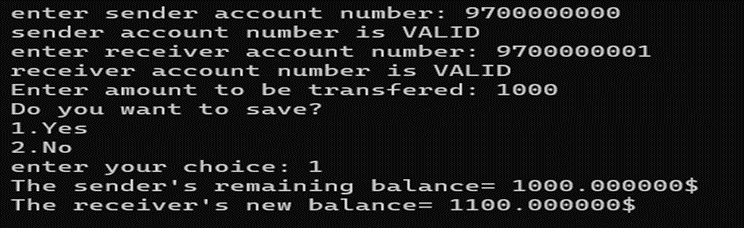
Call the function Account\* load (int \*num\_acc) to return all accounts and their number. Check the existence of the sender’s account by a for loop for 0 to numAccounts. Check the existence of the receiver’s account by a for loop for 0 to numAccounts. If the sender’s account and receiver’s account are found, the program checks the transfer conditions. Asks the user if they want to save the process. If confirmed, decrease the sender’s account balance and increase the receiver’s account balance by the amount and call the function void save (Account \*acc2,int num\_acc) to save the new accounts details in the file “accounts.txt”.

After successfully doing the process, it prints a success message with the sender’s remaining balance and receiver’s new balance. Create or append to a file with the sender’s account number name and print the process done. Closes the file after. Create or append to a file with the receiver’s account number name and print the process done. Closes the file after and frees the allocated memory and returns to the menu() function.

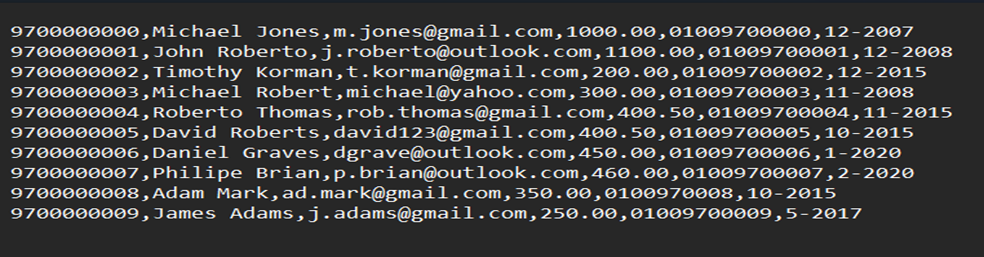
Errors:

* If the entered sender’s account number or receiver’s account number does not exist, prints an error message, and recursively calls the function again.
* If the entered amount is negative or more than the sender’s balance, prints an error message that the transaction is not successful. If the user enters a number not asked in the save question, it prints an error message and loops until the user enters it correctly.

Sample Run:



The “accounts.txt” file after saving the process



The “9700000000.txt” file after saving the process



The “9700000001.txt” file after saving the process



### **REPORT**

Input:

* Account number for report

Mechanism:

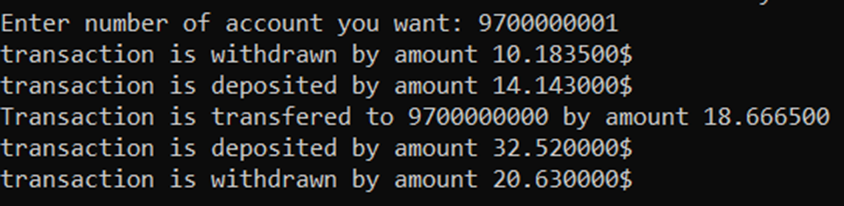
This function asks the user to enter an account number. The function fflush(); is used to clear the buffer before fgets(), strcspn(); is used to remove the new line character at the end of the input.

This function is used to print the last 5 transactions made by the account which its number is entered by the user. It opens the file of the account and reads the transaction line by line and stores it in an array. Then this array is printed using a for loop.

Errors:

The function checks if the account number entered by the user is found or not by using isAccountNumberUnique() function, and if not found, it recalls the report function again.

Sample Run:



### **PRINT SORT**

Input:

* Sorting choice (1 for name, 2 for date, 3 for balance)

Mechanism:

The user is asked to choose how to sort accounts either by name, date, or balance. The user should enter 1 for name, 2 for date, 3 for balance.

The function uses switch () to handle each choice then it calls the function to sort with the type which the user chose, and accounts are returned in the same type of sort. For each function used to sort, it uses bubble sort. The function void Print\_Sort() loads all accounts and gets its number from load(int \*num\_acc) then it prints all accounts sorted using a for loop.

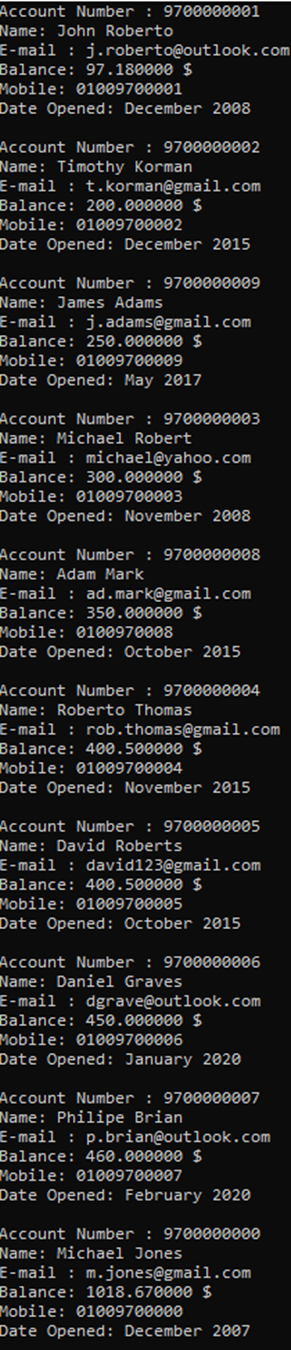
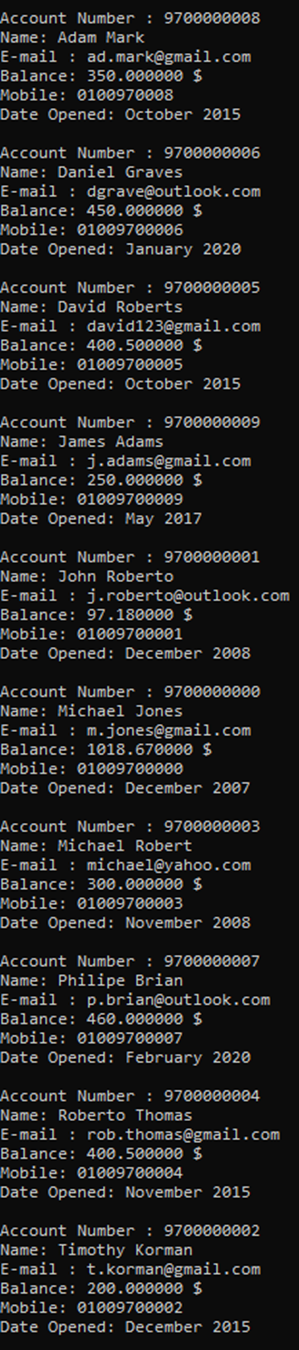
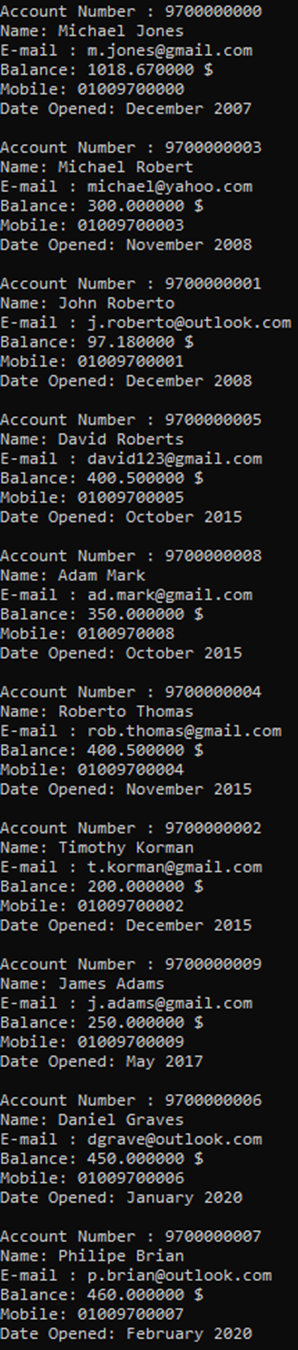
Errors:

If the user entered a not suitable choice: there is a default in switch () that recalls the Print\_Sort() function again.

Pseudocode:

The type of sort used in each function of sort is ascending bubble sort.

First of all it use two for loops to compare the array elements with each other, it compares first element with the second, and if the second is greater than the first they are swapped. This step is repeated till it is compared with all other elements of the array. By the end of the first internal loop (when j=size-2) the biggest value is at the end of the array, So we don’t need to loop to the end of the array again because it is already sorted. These steps are repeated till we reach the end of the external array.

Sample Run:

### **SAVE**

Input:

* In any function that needs to save data the user is asked to choose whether he wants to save or not if yes void save() is called.

Mechanism:

The function take two parameters pointer to an array of Account and number of accounts. First the file we want to save in it is opened in write mode, then we make a for loop till the number of accounts, within each loop accounts are printed with all its data.

Errors:

At the end of the loop to avoid printing a new line in the file we make if condition to check if this is the last account to be printed or not.

A screen shot of a computer

Description automatically generated

**USER MANUAL:**

**1.login:**

takes the username and password form user.

by choosing 1 from the menu then enter the

username and password.

**2.quit:**

If you want to exit the program, choose quit function.

A screenshot of a computer

Description automatically generated

A computer screen shot of white text

Description automatically generated**3.Add:**

Requests user input for new account details (number, name, mobile number, email address, balance) If you want to add another contact choose 1 again and so on till you finish all contacts you want to add.

A screenshot of a computer

Description automatically generated

**3.delete**

Asks the user to input an account number to delete.

A screenshot of a computer

Description automatically generatedA computer screen with white text

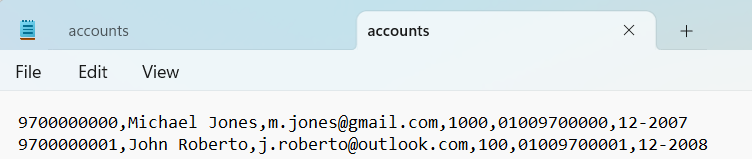
Description automatically generatedchoose 2 from the menu.

A screenshot of a computer

Description automatically generated

**4.modify**

A close-up of a computer screen

Description automatically generatedA computer screen with white text

Description automatically generatedRequests the user to input an account number for modification and if found, allows the user to modify details. choose 3 from the menu.

**5.search**

A screenshot of a computer

Description automatically generatedA computer screen with white text

Description automatically generatedRequests the user to input an account number for querying. Utilizes a linear search algorithm to find the account. choose 4 from the menu.

**6.Advanced Search**

A screenshot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generatedRequests the user to input a keyword for advanced search. Utilizes a linear search algorithm and prints details of found accounts. choose 5 from the menu.

7.Withdraw

A screenshot of a computer

Description automatically generated**A screenshot of a computer

Description automatically generatedA computer screen with white text

Description automatically generated**A screenshot of a computer

Description automatically generatedThe user should be prompted for account number. Then the amount needs to be withdrawn and asks the user if they want to save the process. If confirmed, decrease the account’s balance by the amount. choose 6 from the menu.

**8.deposit**

The user should be prompted for account number. Then the amount needs

A computer screen shot of a computer code

Description automatically generated to be deposited to that account. choose 7 from the menu.

A computer screen with white text

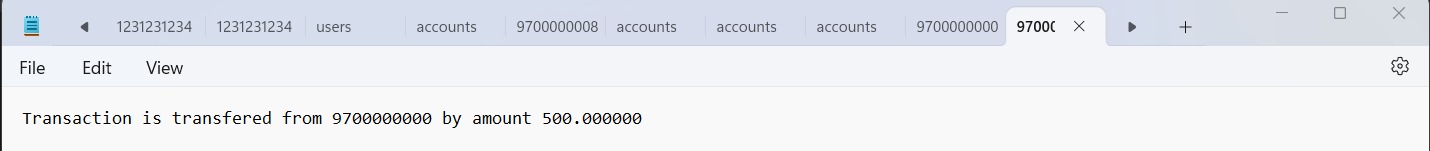
Description automatically generatedA close up of a computer screen

Description automatically generated

**9.transfer**

A computer screen with white text

Description automatically generatedA computer screen shot of a computer code

Description automatically generatedA screenshot of a computer

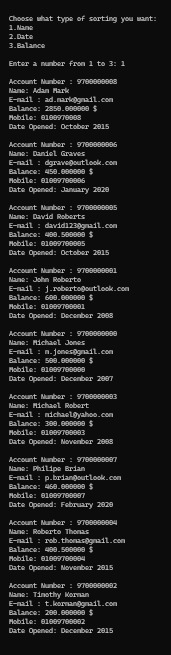
Description automatically generatedThe user transfer money from account to another account. The user prompted for account numbers of sender and receiver. choose 8 from the menu.

**10.report**

A computer screen with white text

Description automatically generatedA screenshot of a computer

Description automatically generatedThe user prompted account number then the prints the last 5 transactions made on this account. choose 9 from the menu.



**11.print(sort)**

Print the data of all accounts, in sorted order. The user is asked to choose how to sort accounts either by name, date, or balance. choose 10 from the menu.

A screenshot of a computer

Description automatically generated