Programming Project Report

Bank management system

Eyad Ahmed Aly -9073

Hazem Hassan Aly-8619

Adham Ahmed-8640

Eyad Mohamed Moawad-8744

Work Description

1.Objective:

The primary objective of this code is to manage client accounts, allowing users to perform various operations like adding, modifying, deleting account,searching for account,withdrawl and depositing while maintaining data integrity.

2.Functionalities Implemented:

2.1Addition of Client Accounts:

Users can input and add new client accounts.

Account details including account number, name, email, balance, phone number, and creation date are captured and stored securely

2.2Modification of Account Information:

Provides functionality to modify existing account information.

Enables users to update account details such as name, email, and phone number while maintaining data consistency.

2.3 Deletion of Accounts:

Allows deletion of accounts with a zero balance.

Ensures data integrity by restricting the removal of accounts with non-zero balances.

2.4 Reporting Feature:

Includes functionality to generate reports that saves and prints the last 5 transactions made on this account

2.5 Search and Advanced Search:

Provides search functionalities to find specific accounts by account number

Advanced search allows for more refined searches based on multiple criteria like balance, date, or other account attributes.

2.6 Deposit and Withdrawal:

Allows clients to deposit funds into their accounts.

Enables withdrawal of funds while ensuring appropriate balance validation.

2.7 Transfer Funds:

Facilitates fund transfers between accounts, ensuring accurate transaction handling and balance updates.

2.8 Sorting and printing Data:

Implements sorting mechanisms for account data by name, date, and balance.

Sorts accounts alphabetically by name (sortByName).

Sorts accounts by date (sortByDate) and balance (sortByBalance), facilitating easy retrieval of information in various orders.

3.Challenges Overcome:

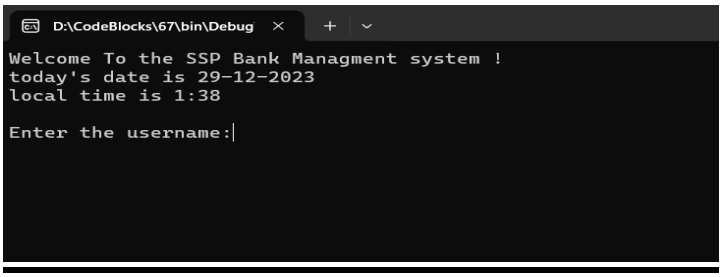
Input Validation: Ensured strict validation of user-provided data to prevent incorrect entries.

Data Consistency: Implemented checks and balances to maintain data consistency during account modifications and deletions.

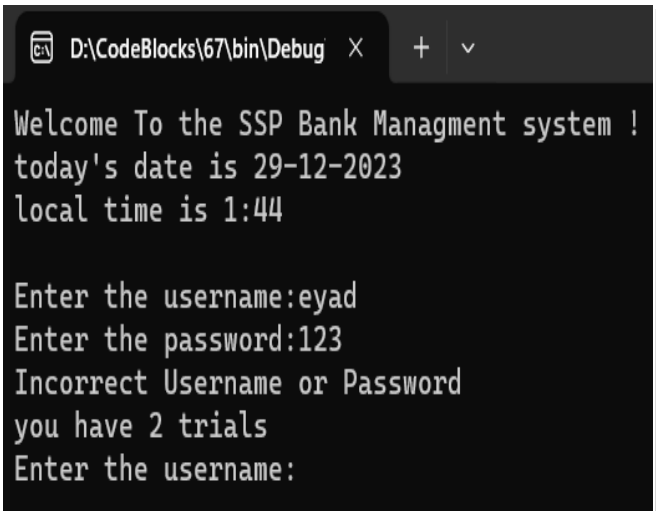
File Operations: Managed file read/write operations effectively to prevent data loss or corruption.

Sample Runs

When you first run the program, it asks to enter the username and the password of the user, showing a welcome message,date and time as shown below



If you enter the correct username and password it will enter the menu to choose the service you want with a welcoming message and if you entered wrong credentials, it will give you 2 other trials before it shuts down the whole system



Now you will have 11 function to choose from as shown in the menu

0.Menu function

1. Start:
2. Display the list of available services with corresponding numbers for selection.
3. Read user input for the desired service (x).
4. Based on the selected service (x):
    - Utilize a switch statement to execute different functionalities:
        - Case 1: ADD
            - Call the add function to add a new account.
        - Case 2: Delete
            - Call the deleteAccount function to remove an existing account.
        - Case 3: Modify

        - Call the modifyAccount function to modify an account's details.
    - Case 4: Search
        - Call the search function to find an account based on account number.
    - Case 5: Advanced Search
        - Call the advancedSearch function to search by name across accounts.
    - Case 6: Withdraw
        - Call the withdraw function to withdraw funds from an account.
    - Case 7: Deposit
        - Call the deposit function to deposit funds into an account.
    - Case 8: Transfer
        - Call the transfer function to transfer funds between accounts.
    - Case 9: Report
        - Call the report function to display transaction history for an account.
    - Case 10: Print
        - Call the print function to display account information based on sorting criteria.
    - Case 11: Quit
        - Terminate the program.
    - Default:
        - Display "Invalid choice" message for any other input.
5. Loop completed, end the function.

Sample run

```
1.ADD
2.Delete
3.Modify
4.Search
5.Advanced Search
6.Withdraw
7.Deposit
8.Transfer
9.Report
10.Print
11.quit
```

1.Login Function

1. Initialize Variables:
    - Initialize variables for counters (i, j) and a flag to track successful login attempts.
2. Welcome Message:
    - Display a welcome message introducing the SSP Bank Management system.
    - Retrieve and display the current date and local time.
3. Authentication Loop:
    - Begin a loop allowing three login attempts (j < 3).
    - Request the user to input a username and password.
    - Compare the input credentials with those stored in the log array (d).
        - Iterate through the log array (i < load_Login(d)) to check each stored username and password.
        - If a match is found:
            - Display a success message.
            - Display a personalized welcome message for the logged-in user.
            - Return a success code (1) to indicate successful login.
4. Handling Incorrect Login Attempts:
    - If the username or password is incorrect within the allowed attempts:
        - For each incorrect attempt, display a message indicating the remaining attempts.
        - If three incorrect attempts are made:
            - Display a failure message for exceeding login trials.
            - Exit the program.
5. Final Message:
    - If login fails due to trial limit:
        - Display a message indicating login failure due to exceeded trials.
    - Return a failure code (0) to indicate unsuccessful login.

Sample runs

```
Welcome To the SSP Bank Managment system !
today's date is 29-12-2023
local time is 2:36

Enter the username:eyad
Enter the password:123@
login success.
Welcome eyad!
```

```
Enter the username:eyad
Enter the password:123
Incorrect Username or Password
you have 2 trials
Enter the username:eyad
Enter the password:1
Incorrect Username or Password
you have 1 trials
Enter the username:eyad
Enter the password:2
Login failed. Exiting...
```

2.Add Function

The add function allows the user to insert an account to the function and be prompted to add the details of the new account , the mechanism of this function is as the following:

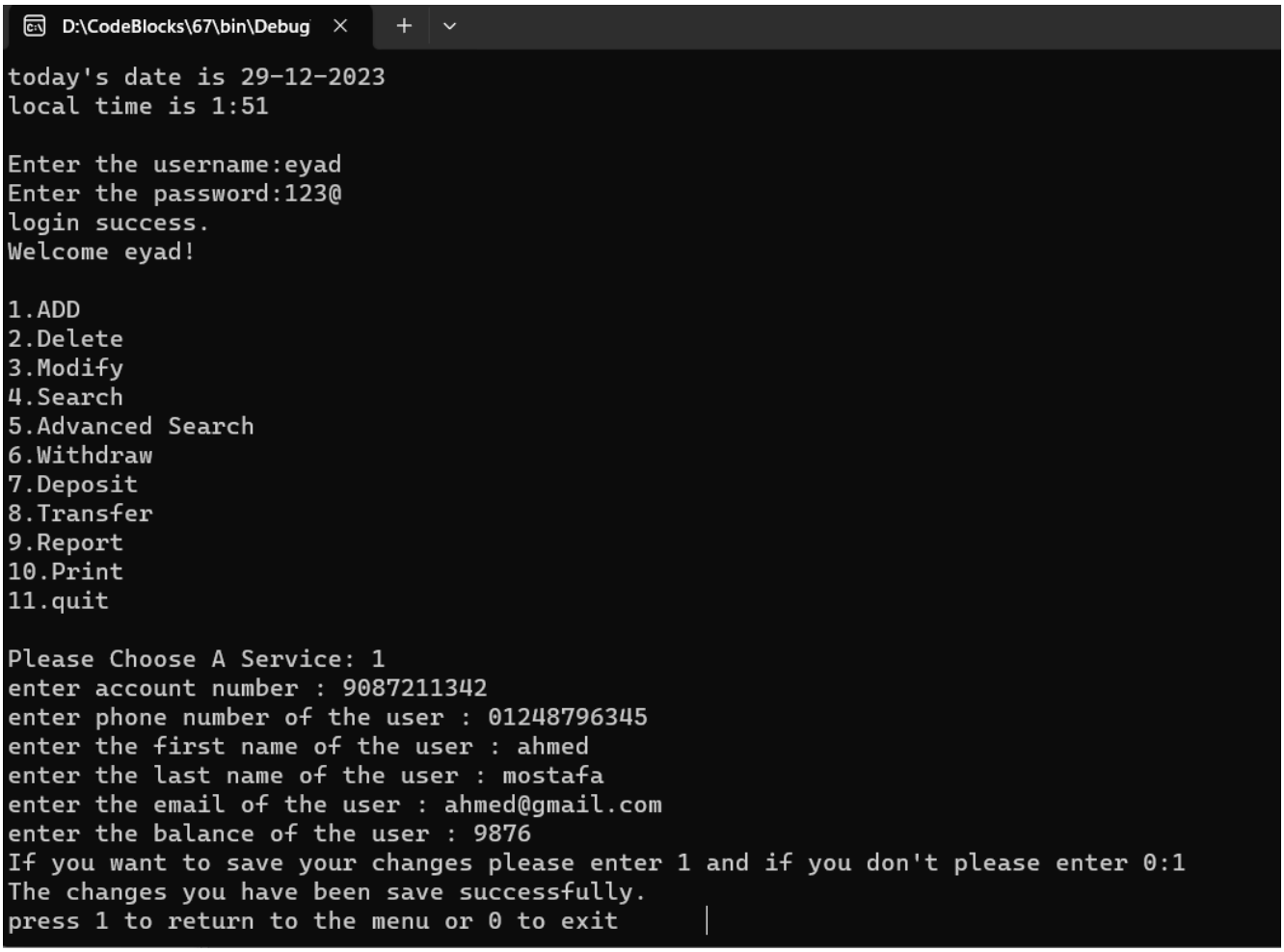
algorithm

1. **Start:**
2. Read and validate the account number:
    - Ensure the entered account number is unique in the system.
    - Prompt the user to enter a different account number if it already exists.
    - If the entered account number is invalid (e.g., incorrect format), request a valid one.
3. Validate the phone number format:
    - Ensure the entered phone number adheres to the expected format (e.g., 11 digits).
    - Prompt the user to enter a valid phone number if it doesn’t match the format.
4. Gather client information:
    - Obtain and store client details such as first name, last name, email, and balance.
5. Capture the current date and time:
    - Record the current date and time to mark the account creation.
6. Create a new entry in the client data structure:
    - Assign the entered account number to the new account.
    - Populate the fields with the collected client information (name, email, balance, etc.).

- Set the creation date and time for the account.

7. Save the updated client data:
    - Persist the updated client data, including the new entry, to the data storage (e.g., file).
8. Associate a file with the account number:
    - Generate a filename based on the account number.
    - Create a file with this filename to associate additional information if necessary.

And the sample runs of this function as following:



3.Deletion Function

The deletion function allows the user to delete the wanted account(balance must be zero)

And its algorithm as following

Algorithm

1. Start:
2. Receive the account number (se) to be deleted from the user.
3. Validate the input account number:

- Check if the entered account number is in a valid format.
- Ensure the account number exists in the system.

4. Loop through the client data array to find the account:
    - Search for the account with the matching account number (se).
5. If the account is found:
    - Check if the account balance is zero:
        - If the balance is zero, proceed with deletion.
        - If the balance is greater than zero, exit and notify the user that the deletion is not allowed due to a non-zero balance.
6. Delete the account:
    - Remove the account from the client data array (d) by shifting the elements.
    - Decrement the count of total accounts (c).
7. Remove the associated file:
    - Generate the filename based on the account number.
    - Use the remove function to delete the associated file.
8. Save the updated client data:
    - Update the stored client data after account deletion.

And sample runs as following:

```
enter account number that you want to delete: 0000000000
Account found!
If you want to save your changes please enter 1 and if you don't please enter 0:1
The changes you have been save successfully.
Account deleted successfully.
press 1 to return to the menu or 0 to exit
```

```
enter account number that you want to delete: 9087211342
Account found!
Balance is greater than zero.
press 1 to return to the menu or 0 to exit
```

4.Modify Account

This function allows you to modi name,email or phone number

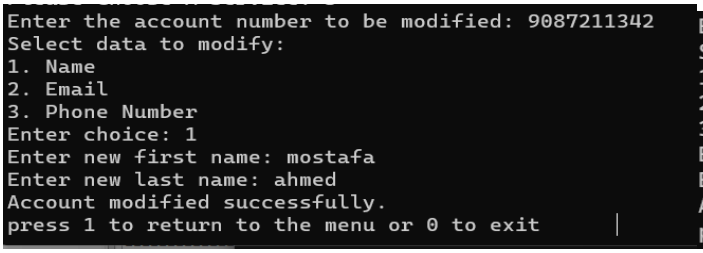
Algorithm:

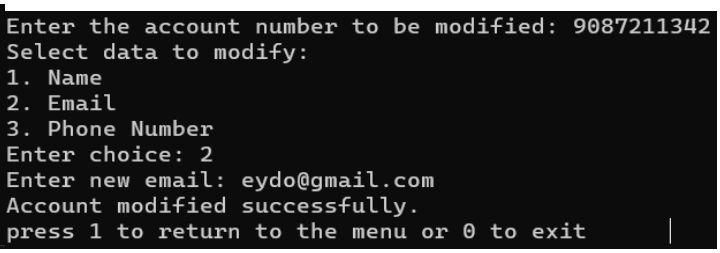
1. **Start:**
2. Request the account number (**accToModify**) and the data to be modified (**choice**) from the user.
3. Open the "accounts.txt" file for reading and create a temporary file ("temp_accounts.txt") for writing.
4. Define a structure (**Account**) to store account details.

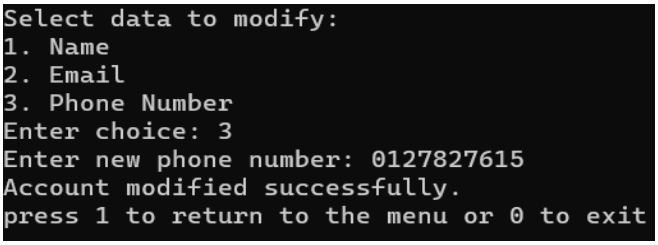
5. Initialize a flag (**found**) to determine if the account to modify is found in the file.
6. Loop through the file content:
    - Read account details line by line.
    - Check if the account number matches the provided account number (**accToModify**).
    - If found:
        - Modify the selected data based on the user's choice:
            - Update the first name and last name if **choice** is 1.
            - Update the email if **choice** is 2.
            - Update the phone number if **choice** is 3.
        - Write the modified or unmodified account details to the temporary file.
7. Close both the input and temporary files.
8. Remove the original "accounts.txt" file and rename the temporary file to "accounts.txt".
9. If the account is found:
    - Print a success message confirming the modification.
10. If the account is not found:

- Print a message indicating that the account was not found, and the modification was rejected.

Sample runs:







5.Search

This function allows you to search and show data for account from the account number

Algorithm

1. Start:
2. Prompt the user to enter the account number (se) they want to search.
3. Initialize a counter variable k to track the number of occurrences of the searched account number.
4. Loop through the client data array:

- Check if the account number at index i matches the entered account number (se).
- If found:
    - Increment the counter k.
    - Display the account details:
        - Account number.
        - Name (first and second name).
        - Email.
        - Mobile number.
        - Balance.
        - Date (month and year).

5. If no matching account number is found (k==0):
    - Print a message indicating that no account with that number was found

Sample runs:

```
Enter the account number that you want to search: 9087211342
Account number: 9087211342
Name: ahmed mostafa
Email: ahmed@gmail.com
Mobile: 01248796345
Balance: 9876.00$
Date: december 2023
```

6.Advanced Search

The user must supply a keyword, and the system should provide all data for all accounts whose name contains that keyword or a message indicating that no matches are found.

Algorithm

1. Start:
2. Prompt the user to enter the name (ad) they want to search.
3. Loop through the client data array:
    - Check if the entered name matches either the first name or the second name at index i.
    - If a match is found:

- Display the account details:
    - Account number.
    - Name (first and second name).
    - Email.
    - Mobile number.
    - Balance.
    - Date (month and year).

Sample runs

```
Enter the name that you want to search: mostafa
Account number: 9087211342
Name: ahmed mostafa
Email: ahmed@gmail.com
Mobile: 01248796345
Balance: 9876.00$
Date: december 2023

press 1 to return to the menu or 0 to exit
```

7.withdrawl

1. Start:
2. Prompt the user to enter the account number (account_number) from which they want to withdraw.
3. Validate the entered account number:
    - Ensure the account number is valid.
4. Find the account index based on the entered account number.
5. If the account is found:
    - Prompt the user to enter the withdrawal amount (amount_str).
    - Validate the entered amount:
        - Ensure it's a valid float value.
        - Check if the amount is within the acceptable range (greater than zero and not exceeding $10,000).
    - If the entered amount is invalid, prompt the user for a valid amount.
6. Check if the account balance is sufficient for the withdrawal:
    - If the balance is less than the withdrawal amount, prompt the user for a valid withdrawal amount or restart the process.
7. If the withdrawal is valid:
    - Deduct the withdrawal amount from the account balance (x[accIndex].balance).

- Display the successful transaction message with the new balance.
- Attempt to save the updated account data:
    - If unsuccessful, revert the balance back to its original value.

8. If the account is not found:
    - Display a message indicating that the account number is not found.

Sample runs

```
Enter the account number:9087211342
Please enter the amount to be withdrawn:5487
Transaction is done successfully.
The new balance is 4389.00.
If you want to save your changes please enter 1 and if you don't please enter 0:
```

```
Enter the account number:9087211342
Please enter the amount to be withdrawn:131221
Invalid amount.
If you want to withdraw a larger amount, you can withdraw the amount in batches.
Please enter a valid amount that don't exceed 10000$:
```

8.Deposit

1. Start:
2. Prompt the user to enter the account number (account_number) for the deposit.
3. Validate the entered account number:
    - Ensure the account number is valid.
4. Find the account index based on the entered account number.
5. If the account is found:
    - Prompt the user to enter the deposit amount (amount_str).
    - Validate the entered amount:
        - Ensure it's a valid float value.
        - Check if the amount is within the acceptable range (greater than zero and not exceeding $10,000).
    - If the entered amount is invalid, prompt the user for a valid amount.
6. If the deposit amount is valid:
    - Add the deposit amount to the account balance (x[accIndex].balance).
    - Display the successful transaction message with the new balance.
    - Attempt to save the updated account data:
        - If unsuccessful, revert the balance back to its original value.
7. If the account is not found:
    - Display a message indicating that the account number is not found.

Sample runs:

```
Enter the account number:9087211342
Please enter the amount to be deposited:587
Transaction is done successfully.
The new balance is 4853.00.
If you want to save your changes please enter 1 and if you don't please enter 0:
```

```
Enter the account number:9087211342
Please enter the amount to be deposited:22112121
Invalid amount.
If you want to deposit a larger amount, you can deposit the amount in batches.
Please enter a valid amount that don't exceed 10000$:
```

9.Transfer

1. Start:
2. Prompt the user to enter the valid sender account number (sender).
3. Validate the sender's entered account number:
    - Ensure it's a valid account.
4. Prompt the user to enter the receiver's account number (receiver).
5. Validate the receiver's entered account number:
    - Ensure it's a valid account.
6. Check for the sender and receiver accounts in the array:
    - Loop through the accounts array to find the sender's account (sender) and the receiver's account (receiver).
    - Set flags (found_send and found_rec) based on whether both accounts are found.
7. If either the sender or receiver is not found, display a corresponding error message.
8. If both sender and receiver are found:
    - Prompt the user to enter the amount to be transferred (amount).
    - Check if the sender has enough balance for the transfer:
        - If the sender's balance is sufficient:
            - Deduct the amount from the sender's balance and add it to the receiver's balance.
            - Display the old and new balances for both sender and receiver.
            - Attempt to save the updated account data.
            - If saving is successful:
                - Update the transaction records for sender and receiver.
            - If saving fails:

- Revert the balances back to their original values.

9. If the sender's balance is insufficient, display an error message.

```
Enter valid sender account number: 9087211342
Enter reciver account number: 1231231231
Sender found
reciver found
Enter amount of money: 123
sender's old balance= 9936.00  reciver's old balance = 840.00
sender's new balance= 9813.00  reciver's new balance = 963.00
If you want to save your changes please enter 1 and if you don't please enter 0:
```

```
Enter valid sender account number: 9087211342
Enter reciver account number: 1231231231
Sender found
reciver found
Enter amount of money: 1000000000000
not enough balance in senders account
press 1 to return to the menu or 0 to exit
```

10.Report

1. Start:
2. Prompt the user to enter the account number for the transaction report (accn).
3. Validate the entered account number:
    - Ensure it's a valid account.
4. Concatenate the account number with ".txt" to form the filename.
5. Attempt to open the file associated with the account number in read mode (f).
    - If the file opening fails, display an error message and exit the function.
6. Initialize a counter i for transaction numbering.
7. If the file is empty (no transactions):
    - Display a message indicating no transactions have been made for this account.
8. If the file contains transactions:
    - Reset the file pointer to the beginning (rewind(f)).
    - Read and display up to the first 5 transactions from the file:
        - Read a transaction (transaction) and an amount (amount) from the file.
        - Display the transaction number (i), type of transaction (transaction), and the amount (amount) involved.
        - Increment the transaction number (i).

- Continue reading and displaying transactions until reaching the fifth transaction or the end of file (feof).

Loop completed or no transactions displayed, end the report

Sample runs

```
Please Choose A Service: 9
please enter account number: 1231231231
Transaction #1 was reciving an amount = 123.00$
press 1 to return to the menu or 0 to exit
```

```
please enter account number: 9087211342
Transaction #1 was sending an amount = 123.00$
press 1 to return to the menu or 0 to exit
```

11.Print

1. Start:
2. Prompt the user to enter the sorting criterion:
    - Options are Name, Balance, and Date.
    - Read the user's input (way).
3. Based on the chosen sorting criterion (way):
    - Utilize a switch statement to execute different sorting functions:
        - Case 1: Sort by Name:
            - Call sortbyname function to sort accounts by name.
            - Print details of each account in the sorted order:
                - Display Account Number, Name, Email, Mobile, Balance, and Date for each account.
        - Case 2: Sort by Balance:
            - Call sortbybalance function to sort accounts by balance.
            - Print details of each account in the sorted order:
                - Display Account Number, Name, Email, Mobile, Balance, and Date for each account.
        - Case 3: Sort by Date:
            - Call sortbydate function to sort accounts by date.
            - Print details of each account in the sorted order:
                - Display Account Number, Name, Email, Mobile, Balance, and Date for each account.
4. Loop completed, end the function.

```
Please Choose A Service: 10
please enter the sorting way:
1)Name.
2)balance.
3)date.
```

```
Account number: 9087211342
Name: ahmed mostafa
Email: ahmed@gmail.com
Mobile: 01248796345
Balance: 9813.00$
Date: 12-2023

Account number: 1231231231
Name: eyad ahmed
Email: sfdvf@gmail.com
Mobile: 12312312313
Balance: 963.00$
Date: 12-2023
```

12.quit

The quit() function is a straightforward function that invokes the exit() system call to terminate the program. When called, it immediately stops the execution of the program and exits, returning control to the operating system.In this specific case, it's named quit() and uses exit(1), which indicates a non-zero status code, typically signaling an abnormal termination of the program. This might be used to signify that the program encountered an unexpected issue before exiting

Algorithms of other important functions

1.sorting

Bubble Sort for Sorting by Name:

This sorting algorithm sorts a list of client structures by their first names.

1. Initialization: The function sortbyname takes an array of client structures (x) and the number of accounts (numberofaccounts) as parameters.
2. Iterative Comparison and Swapping:
    - The outer loop runs from 0 to numberofaccounts - 1 to iterate through the elements of the array.
    - The inner loop also runs from 0 to numberofaccounts - i - 1, where i is the index from the outer loop.
    - Inside the loop, strcasecmp compares the first names of adjacent elements (x[j] and x[j + 1]). If they are out of order (determined by the return value of strcasecmp), they are swapped using a temporary variable (temp).
3. Repetition and Optimization:

- The process repeats for each pair of adjacent elements until the entire array is sorted.

Bubble Sort for Sorting by Balance:

This sorting algorithm sorts the client array based on the balance of the accounts.

The structure of the algorithm is similar to the sorting by name, except the comparison and swapping are based on the balance (x[j].balance and x[j + 1].balance). If the balance of x[j] is greater than x[j + 1], the two elements are swapped.

Bubble Sort for Sorting by Date:

This algorithm sorts the client array based on the date (year and month) of the accounts.

It again follows the same structure as the other bubble sort implementations but involves a more complex comparison between dates (x[j].d.year, x[j + 1].d.year, x[j].d.month, x[j + 1].d.month). If the years are out of order, or if the years are the same and the months are out of order, the elements are swapped accordingly.

5.Save

1. **Input Validation:**
    - Prompt the user to indicate if they want to save changes.
    - Read the user's input and store it in the variable **valid**.
2. **Saving Changes (if requested):**
    - If **valid** is true (1):
        - Attempt to open the "accounts.txt" file in write mode (**"w"**).
        - If the file opening fails:
            - Print an error message indicating the issue.
            - Return 0 to signal a failure in saving changes.
        - If the file opens successfully:
            - Iterate through each account in the **x** array using a loop (from 0 to **numofaccounts - 1**).
            - For each account:
                - Write its details into the file using **fprintf**.
            - Print a success message confirming that the changes have been saved.
            - Close the file.
            - Return 1 to indicate successful saving of changes.
3. **No Changes Saved (if not requested):**
    - If the user decides not to save changes (**valid** is false (0)):
        - Print a message indicating that no changes have been saved.
        - Return 0 to indicate that no changes were made.

6.Load

1. Open the File
    - Attempt to open the file named "accounts.txt" in read mode.
    - If the file opening fails:
        - Print an error message.
        - Return 0 to indicate a failure.
2. Read Account Information
    - Initialize an index variable i to 0.
    - While the index i is less than 100 (maximum accounts to read) and the end of the file is not reached:
        - Read each line of the file.
        - Use fscanf to parse and extract data according to the expected format:
            - Read the account number, first name, last name, email, balance, mobile number, month, and year from the file, assuming they are separated by commas.
            - Store the read data into the respective fields of the client structure array at index i.
3. Return the Number of Loaded Accounts
    - Return the value of i, representing the number of successfully read accounts.