# Bank Management System

## 1. Introduction:

### 2.1-Purpose of the System:

The Banking System is designed to manage and perform various operations related to user accounts. Users can perform transactions, inquire about account details, and manage their accounts seamlessly.

### 2.2-Features:

1. User account management
2. Deposit, withdrawal, and transfer of funds
3. Account querying and reporting
4. Sorting and printing options
5. Advanced search capabilities
6. Secure login and logout

### 2.3-System Components:

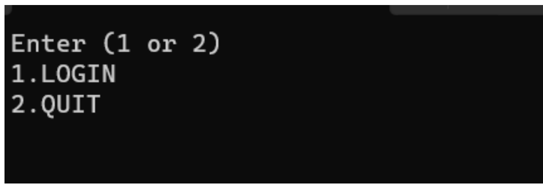
The system comprises user accounts, each with unique identifiers, personal information, and transaction history. Account details are stored in a structured manner, allowing easy retrieval and modification.

## 2. Getting Started

### 2.1 Login:

Access the system by selecting option 1 on the main menu.

Enter your username and password as prompted.

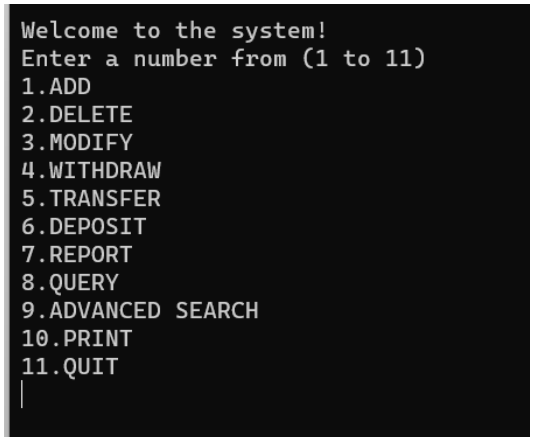


```
Enter (1 or 2)
1.LOGIN
2.QUIT
```

### 2.2 Quitting the System:

Choose option 2 on the main menu to exit the system.

## 3. Main Menu Options:



```
Welcome to the system!
Enter a number from (1 to 11)
1.ADD
2.DELETE
3.MODIFY
4.WITHDRAW
5.TRANSFER
6.DEPOSIT
7.REPORT
8.QUERY
9.ADVANCED SEARCH
10.PRINT
11.QUIT
```

### 3.1 Add an Account:

Select option 1 to add a new account.

Follow the prompts to enter account details.

### 3.2 Delete Account:

Choose option 2 to delete an account.

Input the account number for deletion.

### 3.3 Modify Account

Select option 3 to modify an account.

Enter the account number and follow the prompts for modification.

### 3.4 Withdraw:

Select option 4 for withdrawals.

Enter the account number and the withdrawal amount.

### 3.5 Transfer:

Use option 5  to transfer funds.

Input source and destination account numbers and the transfer amount.

### 3.6 Deposit:

Choose option 6 to deposit funds.

Provide the account number and the amount to deposit.

### 3.7 Reports:

Access transaction reports by choosing option 7.

Enter the account number to view last 5 transactions.

### 3.8 Query Account:

Pick option 8 to inquire about an account.

Enter the account number to display details.

### 3.9 Advanced Search

Select option 9 for advanced search.

Enter a keyword to search for specific account details.

### 3.10 Sort and Print:

Choose option 10 to sort and print account details.

Follow the prompts to select sorting criteria.

### 3.11 Logout and Quit:

Select option 11 to quit the system.

# 4.Validations functions:

We introduced dedicated parameter validation functions to enhance code modularity and readability, promoting a cleaner and more efficient implementation across various sections of the codebase.

## 4.1 checkNumber Function:

**Purpose:** To verify if a given string contains only numerical digits.

**Parameters:** Takes a string as input.

**Steps:** Check if the user input contains only numerical values if not, prints error message.

**Return Value**: Returns 1 if the string consists of numbers only, and 0 otherwise.

## 4.2 checkName:

**Purpose:** Verify whether the given element is comprised of characters.

**Parameters:** Takes a string and index as input.

**Steps:** Check if the user input contains only characters if not, prints error message.

**Return Value**: Returns 1 if the string consists of characters only, and 0 otherwise.

## 4.3 validateAccountNumber:

**Purpose**: The validateAccountNumber function scans and verifies the user-entered account number. It checks if the account number consists of numbers only, confirms the existence of the account number, and validates its length.

**Parameters:** Takes a variable, "printvalue," as the displayed user message and "i" as a pointer/index.

**Steps:**

1-Scans the account number from the user.

2-Checks if the entered account number consists of numbers only (using checkNumber).

3-Checks the existence of the account number; if not, prints an error message.

4-Validates the length of the account number (exactly 10 digits) if not , prints error message.

**Return Value**: Returns the validated account number.

## 4.4 validateAccountDuplication:

**Purpose**: The validateDuplication function scans and verifies the user-entered account number. It checks if the account number consists of numbers only, confirms the inexistence of the account number, and validates its length.

**Parameters:** Takes a variable, "printvalue," as the displayed user message and "i" as a pointer/index.

**Steps:**

1-Scans the account number from the user.

2-Checks if the entered account number consists of numbers only (using checkNumber).

3-Checks the existence of the account number; if exists, prints an error message.

4-Validates the length of the account number (exactly 10 digits) if not , prints error message.

**Return Value**: Returns the validated account number.

## 4.5 validateName:

**Purpose:** The ‘validateName’ function scans and validates a name entered by the user. It ensures that the given name comprises only characters and includes both the first and second names.

**Parameters:** Takes no additional parameters

**Steps:**

1-Scans the name from the user.

2-Checks if the entered name consists of characters only (using checkName).

3-Verifies that the name includes both the first and second names if not, prints error message.

**Return Value**: Returns the validated name.

## 4.6 validateEmail:

**Purpose:** The ‘validateEmail‘ function scans and validates an email entered by the user. It checks for the presence of '@' and '.', ensuring there are characters before and after each symbol.

**Parameters:** Takes no additional parameters.

**Steps:**

1-Scans the email from the user.

2- Checks if the entered email contains '@' and '.' with characters before and after each symbol (using checkName) and prints error message if any of the conditions wasn't verified.

**Return Value**: Returns the validated email.

## 4.7 validateMobile:

**Purpose:** The 'validateMobile' function scans and validates a mobile number entered by the user. It checks that the given mobile number consists only of numbers and has a length of exactly 11 digits.

**Parameters:** Takes no additional parameters.

**Steps:**

1- Scans the mobile from the user.

2- Checks if the entered mobile number consists of numbers only (using checkNumber).

3- Verifies that the length of the mobile number is exactly 11 digits if not, prints error message.

**Return Value**: Returns the validated mobile number.

## 4.8 validateBalance:

**Purpose:** The 'validateBalance' function scans and validates a balance entered by the user. It checks if the balance consists only of numbers.

**Parameters:** Takes no additional parameters.

**Steps:**

1- Scans the balance from the user.

2- Checks if the entered balance consists of numbers only (using checkNumber).

**Return Value**: Returns the validated balance.

## 4.9 validateBalance10000:

**Purpose:** The ‘validateBalance10000 ’ function is employed to verify if the provided balance is less than 10000 (the maximum amount per transaction) and to print an error message to the user.

**Parameters:** Takes balance as input.

**Steps:** Checks if the given balance is smaller than 10000 if not, prints error message.

**Return Value**: Returns 1 if the balance is greater than 10000; otherwise, returns 0.

### 4.10 validateBalanceExistance:

**Purpose:** The ‘validateBalanceExistance’ function is employed to determine if the user has sufficient balance in their account to complete a transaction and to print an error message to the user if necessary.

**Parameters:** Takes balance as input.

**Steps:** Checks if the user has enough balance to complete the transaction if not, prints error message.

**Return Value**: Returns 1 if the user has enough balance; otherwise, returns 0.

### 4.11 askMenu:

**Purpose:** The askMenu function is invoked when an incorrect account number that doesn’t exist or an invalid balance is entered by the user. It prompts the user to decide whether to continue editing or return to the menu.

**Parameters:** Takes no additional parameters.

**Steps:** Asks the user to decide if they want to continue editing or return to the menu.

**Return Value**: The function interacts with the user to determine the desired course of action without returning a specific value.

### 4.12 askSave:

**Purpose:** The ‘askSave’ function is invoked following every user modification, prompting the user to decide whether to save or discard the changes.

**Parameters:** Takes no additional parameters.

**Steps:** Asks the user to choose between saving or discarding the changes.

**Return Value**: Returns 1 if the user wants to save the changes; otherwise, it returns 0.

## 5.Main Functions:

## 5.1 MENU:

The menu function is responsible for displaying the available options to the user and receiving their input. It provides a clear and concise list of functions that the user can choose from. Each function is assigned a number, and the user is prompted to enter the corresponding number to select the desired operation. The menu function ensures user-friendly interaction with the banking system.

**Inputs:**

-The index of the selected function.

**Error handling:**

Prints message error if the user enters a number that is not in range and keep asking until he gives right answer.

## 5.2 LOGIN:

User is prompted to enter their username and password, which are then compared against the stored usernames and passwords in the "users.txt" file. If a match is found, the user is granted access to the system and their account data is loaded. In case of invalid credentials, an appropriate error message is displayed.

**Inputs:**

- Username: The user enters their username.

- Password: The user enters their password.

**Error Handling:**

Invalid username or password: If the entered username or password does not match the stored credentials, an error message is displayed to the user, indicating that the login attempt failed. This provides security by preventing unauthorized access to the system.

## 5.3 LOAD:

The load function enables the retrieval of account information from the "accounts.txt" file. After successful login, the program opens the file in read mode and reads each line, parsing the comma-separated values to extract the account details such as account number, name, email, balance, mobile number, and opening date. The data is then stored in a suitable data structure, such as an array or linked list. Error handling is implemented to detect and display error messages for any issues encountered during file handling or parsing.

**Inputs:**

- None

**Error Handling:**

- File not found: If the "accounts.txt" file is not found or cannot be opened, an appropriate error message is displayed to the user, indicating that the account information could not be loaded. This ensures that the program can handle file-related errors gracefully and provide feedback to the user.

## 5.4 ADD:

The function is made so the user would be able to create a new account , save it into the file “accounts.txt” and to create a file with the account number as its name.it asks the user for its name, email, mobile, balance and stores automatically the time of opening the account after checking that the account number doesn’t exist already in the system and it calls all the validation functions for all the previous demands from the user and it frees the permanent memory from the variables stored in it.

**Inputs:**

-Account Number

-Name

-Email

-Mobile Number

-Balance

**Error handling:**

If the file for the account is not opened and the pointer returns null then a message is printed informing the user of the error.

## 5.5 DELETE:

The function gives the user the right to delete an account only if its balance is equal to 0 and it removes the file created with its account number and calls couple of functions that ask the user if he want to save changes and another that validates the account number and the function delete frees the memory from the account number saved in it.

**Inputs:**

-Account Number

**Error handling:**

The error message that displays that account number is not equal to zero so the account can’t be deleted ,the error of deleting the file.

## 5.6 MODIFY:

It lets the user choose what he wants to change from his name, mobile and e-mail and in each case it asks the user if he wants to save the changes if he does it modifies the user’s account and if he don’t want to save the changes he returns into menu and calls the validation for each case of email, mobile and name and frees the permanently stored information and save the changes to the file “accounts.txt”.

**Inputs:**

-Account Number

-What to modify

**Error handling:**

If the number entered by the user to choose what he wants to modify is not in range it gives an alerting message and asks the user to re-enter a correct number.

## 5.7 WITHDRAWN:

The user is asked to enter an account number that is validated to exist by validateAccountNumber() then he is asked to enter the withdraw amount and it is validated to be only numbers and less than 10000$ with validateBalance() and validateBalance10000().

The user is then asked if the changes should be saved?

No, menu() is called.

Yes, account[i].balance is decreased by the value entered previously by the user (i is known from “validateAccountNumber()”) . And the transaction is saved in the file whose name is the same as the account number entered by the user.

Finally save() is called to save changes done by the function.

**Input:**

Account number.

Amount to be withdrawn.

**Error handling:**

Errors are handled by validateAccountNumber() ,validateBalance() and validateBalance10000.And appropriate errors are printed inside each function.

## 5.8 TRANSFER:

Two account numbers are required to be entered by the user and are validated and there indexes are known by the help of validateAccountNumber().

Then the user is asked to enter amount of transfer the amount is validated to be a number by ValidateBalance() then the account is validated to have enough balance to accept this transfer with validateBalanceExistance(). Finally, transfer amount is validated to be less than 10000 with validateBalance10000().

User is asked if the changes should be saved?

No, menu() is called.

Yes, the transfer amount is subtracted from the first account and added to the second one. Then transactions are saved in the files designated to each account. Finally, Save() is called to save all changes made by the function.

**inputs:**

two account numbers (first, second)

transfer amount

**error handling:**

If the same account number is entered as first and second error message is displayed and the user is asked to enter a different account as the second account number.

Other errors are handled by validateAccountNumber() ,validateBalance() ,validateBalance10000 and validateBalanceExistance().

## 5.9 DEPOSIT:

The user is asked to enter an account number that is validated to exist by validateAccountNumber() then he is asked to enter to enter the deposit value and it is validated to be only numbers and less than 10000$ with validateBalance() and validateBalance10000().

The user is then asked if the changes should be saved?

No, menu() is called.

Yes, account[i].balance (i is known from “validateAccountNumber()”) value is increased by the value entered previously by the user. And transaction is saved in the file whose name is the same as account number entered by the user.

Finally save() is called to save changes done by the function.

**Input:**

Account number

Amount to be deposited

**Error handling:**

Errors are handled by validateAccountNumber() ,validateBalance() and validateBalance10000.

## 5.10 REPORT:

It asks the user to enter the account number then prints the last five transactions made by the account and displays a message in case of no transactions and it prints the transcations made if it is less than 5 transactions .

**Inputs:**

-Account Number

**Error handling :**

The error in opening the file , the error of the account has no transactions and it prints the transactions made if they are less than 5 transactions.

## 5.11 QUERY:

Calls ValidateAccountNumber(discussed in its partition) and after account number is validated and returned, printer(discussed in its partition) is used to print that account

**Input:**

None

**Error handling:**

Uses ValidateAccountNumber to validate that the number actually exist and where it is in accounts[] array and that the user actually entered appropriate account number.

## 5.12 ADVANCED SEARCH:

The user is asked for a keyword to search for. (character case is neglected in this algorithm)

A for loop is used to loop over accounts[](user array that contain all data in the data base)

A for loop inside it is used to loop over each character in account[].name[]

Each character is compared with the first character in the keyword entered by the user

If a character is found to match a for loop inside it is used to loop over the remaining characters in (account[].name[]) and in (keyword) at the same time

Each character that matches increments a variable named “good” so, if “good” = length of “keyword” means that all consecutive characters matched characters of the keyword which implies that the keyword is present inside account[].name[]

This account is then stored in user array called matched[]

After the for loops checked all accounts[]

Matched[] is printed using printer() and for loop

**Inputs:**

-Keyword

**Error handling:**

A keyword maybe present several times in the same account[].name[]. To prevent the same account from being printed several times “hold” is used to store the index of last stored account in matched[] so if the same account has another instance of the same keyword it can be indicated that this account is already stored once.

Index of matched[] is set to -1 at the start of the function if all for loops terminated without  that index being change it indicates no accounts was found to have the keyword entered and a message is displayed for the user to know.

## 5.13 PRINT:

Asks user to choose a number from 1 to 3 indicating type of sort the data should be sorted and printed with. 1) is sorting by name, 2) is sort by balance, 3) sort by date and each function is chosen using a switch.

After the user is asked to enter 1 or 2 that is then passed to each function as “type” and those functions uses “type” as mentioned in their sections.

Then printer() is used to print each value inside the sorted array with the help of for loop.

**Inputs:**

Value from 1 to 3 indicate sorting by?

Value from 1 to 2 to indicate required order in the sort chosen.

**Error handling:**

ValidateNumber() is implemented after the user is asked to enter a value from 1 to 3 and a value from 1 to 2 to validate that value entered is only a number before checking if the number is inside the range.

Also if at any point the user enter a value outside of the specified range an error message will be displayed and the user will be required to enter another value till he enter one inside the range. This is implemented by the help of do while and if conditions.

Default inside the switch handles error of entering number outside the range of 1 to 3 asked at the start of print function.

## 5.14 SortByName:

By implementing bubble sort and strcmp() this function takes a pointer on user array and sort that array according to value of type if type = 1 it will sort the array in alphabetical order of name inside the user array, else it will be sorted in from Z-A.

**Inputs:**

-pointer of user array[] (array to be sorted)

-type (indicates alphabetical or z-a)

**Error handling:**

None

## 5.15 SortByBalance:

By implementing bubble sort this function takes a pointer on user array and sort that array according to value of type if type = 1 it will sort the array from highest balance to the lowest, else it will be sorted from lowest to highest

**Inputs:**

-pointer of user array[] (array to be sorted)

-type (indicates highest to lowest or lowest to hughest)

**Error handling:**

None

## 5.16 datecmp:

Compare two date structes if the first year is larger return 1 if smaller return -1 if equal it compares months if first is larger return 1 if smaller return -1 if equal return zero indicating both are equal

**Input:**

-Two date structs (first, second)

**Error handling:**

None

## 5.17 SortByDate:

by implementing bubble sort and datecmp() this function takes a pointer on user array and sort that array according to value of type if type = 1 it will sort the array according to the oldest acc to the newest, else it will be sorted from newest to oldest

**Inputs:**

-pomter of user array[] (array to be sorted)

-type (indicates oldest to newest or newest to oldest)

**Error handling:**

None

## 5.18 Printer:

Printer is used to print all values of any user struct in order required by the project.

**Inputs:**

-One user struct.

**Error handling:**

None

## 5.19 SAVE:

The save function is responsible for persisting any modifications or additions made to the account information. When triggered, the function opens the "accounts.txt" file in write mode. It iterates through the account data structure and formats each account's details. The formatted data is then written to the file as a new line. Error handling is implemented to detect and display appropriate error messages for any issues encountered during file handling.

**Inputs:**

- None

**Error Handling:**

File write error: If there is an error while writing the account data to the "accounts.txt" file, an error message is displayed to the user, indicating that the data could not be saved. This ensures that any issues with file writing are properly handled and communicated to the user.

## 5.20 QUIT:

The quit function allows the user to gracefully exit the banking system. It can be triggered from either the login screen or the main menu. Depending on the context, different exit codes can be used to indicate the reason for quitting. The function displays a farewell message to the user and terminates the program using the appropriate termination command or function.

**Inputs:**

- None

**Error Handling:**

None