

BANKING SYSTEM

Made By:

Abdelrahman Mostafa Kamel – 8708

Mohammed Hatem Roushdy – 8995

Zeyad Wael ElMorshedi – 8977

Moustafa Mohammed ElNashar - 8577

Basic_Features

- **User Interface:** The system provides a menu-driven interface for users to perform various banking operations, including adding, deleting, and modifying accounts, as well as depositing, withdrawing, and transferring money. The user interface functions, such as `uiStartup` and `ui`, provide a structured way for users to interact with the system.
- **Account Management:** Users can add new accounts, delete existing accounts, and modify account details such as account numbers, names, email addresses, and mobile numbers. The system also supports advanced search functionality for querying account information based on different criteria.
- **Transaction Handling:** The system allows users to deposit money into accounts, withdraw money from accounts, and transfer money between accounts. Additionally, users can generate transaction reports for individual accounts, providing a detailed overview of account activities.
- **File Operations:** The system interacts with files to load and save account and transaction data. It includes functions for reading and writing transaction data to files, as well as validating file access and existence. The `openFileTillValid` function ensures that file operations are performed securely and with proper error handling.

Basic_Features

- **User Authentication:** The system includes a login process that loads user credentials from a file and validates user login information. The `load_credentials` function is responsible for loading and validating user credentials from the specified file.
- **Error Handling:** The system includes error handling for invalid user input and file access errors. Functions such as `getOnHeapTillValid` and `inputTillValid` ensure that user input is validated and errors are appropriately handled.
- **Data Manipulation:** The system includes functions for manipulating data, such as converting strings to lowercase for accurate comparison, validating the format of data read from files, and ensuring the correctness of user-provided input.
- **Memory Management:** The code includes functions for allocating and freeing memory for different data structures, ensuring efficient memory usage and preventing memory leaks.
- **Portability:** The code is designed to be portable and can work on any operating system, providing flexibility and accessibility to users across different platforms

user interface

uiStartup:

The **uiStartup** function initializes the user interface and handles the initial login process. It sets the stage for users to interact with the banking system by presenting the necessary prompts and options for authentication and access.

ui:

The **ui** function serves as the main user interface, offering a menu-driven approach for users to perform various banking operations. It presents a range of options, such as adding, deleting, and modifying accounts, as well as depositing, withdrawing, and transferring money. This function guides users through the available functionalities and facilitates their interaction with the system.

Account Management

addAccount:

The **addAccount** function allows users to create new accounts by providing necessary details such as account number, name, email address, and other relevant information. It ensures that the new account is properly initialized and added to the system.

deleteAccount:

The **deleteAccount** function enables users to remove existing accounts from the system. It prompts users to specify the account to be deleted and performs the necessary operations to remove the account while maintaining data integrity.

modifyAccount:

The **modifyAccount** function facilitates the modification of account details. Users can update information such as account numbers, names, email addresses, and other account-related data. This function ensures that the modified information is correctly applied to the respective account.

searchAccount:

The **searchAccount** function allows users to search for specific accounts based on various criteria such as account number, name, or other attributes. It provides a mechanism for retrieving account information based on user-defined search parameters.

Transactions Handling

deposit:

The **deposit** function allows users to deposit money into their accounts. It facilitates the addition of funds to the specified account and ensures that the account balance is appropriately updated.

withdraw:

The **withdraw** function enables users to withdraw money from their accounts. It validates the withdrawal amount against the available balance and processes the transaction while updating the account balance accordingly.

transfer:

The **transfer** function facilitates the transfer of funds between accounts. It validates the transfer amount and source account balance, performs the necessary debit and credit operations, and updates the balances of both the source and destination accounts.

generateTransactionReport:

The **generateTransactionReport** function allows users to obtain transaction reports for individual accounts. It retrieves and presents a detailed overview of the account's transaction history, including deposits, withdrawals, and transfers.

© 2023
All rights reserved.

File Operations

openFileTillValid:

The **openFileTillValid** function is responsible for securely opening a file, ensuring that the file operations are performed with proper error handling and validation. It sets the stage for subsequent file-related operations within the system.

load:

The **load** function reads account information from a file, parsing the data and populating the system with the relevant account details. It ensures that the data is loaded correctly and adheres to the expected format.

save:

The **save** function writes account and transaction data to a file, ensuring that the information is stored accurately and consistently. It handles the persistence of data to maintain the integrity of the banking system's records.

appendTransaction:

The **appendTransaction** function adds transaction data to a file, allowing for the recording of financial activities such as deposits, withdrawals, and transfers. It appends new transaction records while maintaining the existing data within the file.

makeTransactionFile:

The **makeTransactionFile** function creates a new transaction file, establishing a structured file for recording transaction data. It ensures that the file is properly initialized and ready to store transaction information.

User Authentication

validateUser:

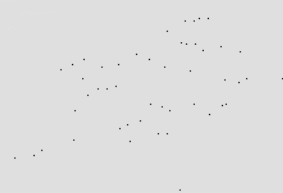
The **validateUser** function verifies the provided username and password against the stored credentials. It ensures that the user is authenticated based on the correctness of the login credentials.

loadCredentials:

The **loadCredentials** function retrieves user credentials from a file, allowing for the validation of login information. It provides the necessary data for authenticating users and controlling access to the banking system.

changePassword:

The **changePassword** function enables users to update their login passwords. It validates the new password and ensures that the user's credentials are securely modified.



Validations

- **Input Validation:** Various functions such as `isValidName`, `isMobile`, and `isValidEmail` are used to validate user-provided input for names, mobile numbers, and email addresses. These functions check for the presence of valid characters, proper formatting, and length constraints to ensure that the input meets the required criteria.
- **Numeric Validation:** The system employs `isPositiveDouble` to validate positive double-precision numbers, ensuring that only valid numerical values are accepted for specific operations, such as depositing or withdrawing money.
- **Account Number Validation:** The `isValidAccNum` function combines numeric validation with a search for existing account numbers, ensuring that entered account numbers are both properly formatted and correspond to existing accounts.
- **Password Validation:** The `isValidUsernamePass` function checks for the presence of spaces in usernames or passwords, enforcing a constraint that disallows spaces in these fields for security reasons.
- **File Format Validation:** When interacting with files, the code checks for proper formatting and integrity. For example, the code verifies that a newline character is present at the end of a file, ensuring that the file format is consistent and complete.

Input_Functions

getnumberTillValid:

This function is designed to obtain numerical input from the user and validate it against specified conditions. It ensures that the input is a valid number and meets any additional criteria, such as being within a certain range.

inputTillValid:

The inputTillValid function repeatedly prompts the user for input until valid data is provided. It handles various types of input and ensures that the entered data meets specific requirements, such as length constraints and format validation.

getOnHeapTillValid:

This function allocates memory on the heap for user input and ensures that the input is valid. It is particularly useful for handling input of variable length, such as strings, while maintaining validation and error handling capabilities.

finput:

The finput function reads input from a file and performs validation to ensure that the input is properly formatted and meets the required criteria. This is crucial for securely handling file input and maintaining data integrity.

User_Manual

1. Gate Menu:

•To open the program, click on its icon. Upon successful execution, you will encounter the "Gate Menu."

•In the "Gate Menu," you will see the following options:

```
1-LOGIN
2-QUIT
Enter option:
```

- Option 1: Login
 - Select this option to log in using your valid username and password.
 - Upon successful login, you will proceed to the main menu.
- Option 2: Quit
 - Select this option to exit the program.

```
Thank you for choosing our services
```

2. Main Menu:

•After logging in, you will reach the "Main Menu."

```
1-ADD
2-DELETE
3-MODIFY
4-SEARCH
5-ADVANCED SEARCH
6-WITHDRAW
7-DEPOSIT
8-TRANSFER
9-REPORT
10-PRINT
11-QUIT
Enter option: _
```


User_Manual

•In the "Main Menu," you will find the following options:

- Option 1: Add a New Account

Use this option to create a new bank account. Each account must have a unique account number.

```
Account Holder name: name
Email: name@mail.com
please enter account number: 9700000010
balance: 1000
Mobile number: 01234567890
Save changes? (y/n) :
```

- Option 2: Delete an Existing Account

Select this option to remove an existing bank account from the system.

```
please enter account number: 9700000005
Save changes? (y/n) : _
```

- Option 3: Modify an Existing Account

Use this option to make changes to an existing bank account, such as updating name, mobile number and email address.

```
please enter account number: 9700000005
Enter new name: name
Enter new email: name@mail.com
Enter new mobile number: 01234567890
Save changes? (y/n) : _
```

- Option 4: Search

This option allows you to search for a specific bank account linear search algorithm.

```
please enter account number: 9700000005
Account Number : 9700000005
Name: David Roberts
E-mail: david123@gmail.com
Balance: 0.00 $
Mobile: 01009700005
Opened: October 2015
```

User_Manual

- Option 5: Advanced Search

Select this option for advanced search functionality, enabling more specific and detailed account searches.

```
Enter keyword: rob
-----
Account Number : 9700000005
Name: David Roberts
E-mail: david123@gmail.com
Balance: 0.00 $
Mobile: 01009700005
Opened: October 2015
-----
Account Number : 9700000001
Name: John Roberto
E-mail: j.roberto@outlook.com
Balance: 94.00 $
Mobile: 01009700001
Opened: December 2008
-----
Account Number : 9700000003
Name: Michael Robert
E-mail: michael@yahoo.com
Balance: 300.00 $
Mobile: 01009700003
Opened: November 2008
-----
Account Number : 9700000004
Name: Roberto Thomas
E-mail: rob.thomas@gmail.com
Balance: 400.50 $
Mobile: 01009700004
Opened: November 2015
-----
```

- Option 6: Withdraw

Use this option to withdraw funds from a selected bank account.

```
please enter account number: 9700000005
How much to withdraw: 100_
```

- Option 7: Deposit

Select this option to deposit funds into a chosen bank account.

```
please enter account number: 9700000005
How much to deposit: 1000
Save changes? (y/n) :
```


User_Manual

- Option 8: Transfer

This option enables you to transfer funds between different bank accounts.

```
Sender:-
please enter account number: 9700000001
Reciever:-
please enter account number: 9700000005
enter amount to be transferred: 10
Save changes? (y/n) :
```

- Option 9: Report

Use this option to generate reports about account transaction histories.

```
please enter account number: 9700000001

-----
withdrawing 1.00
withdrawing 1.00
withdrawing 1.00
withdrawing 1.00
withdrawing 1.00
-----
```

- Option 10: Print

Select this option to print accounts information sorted by name , balance or date using quick sort algorithm in a descending or ascending order according to the user need .

```
1- by name
2- by date
3- by balance
choose how to sort the accounts :1
do you want the sorting to be ascending ? (y/n) : n
-----
Account Number : 9700000001
Name: Timothy Norman
E-mail: t.norman@gmail.com
Balance: 200.00 $
Mobile: 01009700002
Opened: December 2018
-----
Account Number : 9700000004
Name: Roberto Thomas
E-mail: rob.thomas@gmail.com
Balance: 400.50 $
Mobile: 01009700004
Opened: November 2015
-----
Account Number : 9700000002
Name: Philip Brian
E-mail: p.brian@outlook.com
Balance: 400.00 $
Mobile: 01009700007
Opened: February 2016
-----
Account Number : 9700000001
Name: Michael Robert
E-mail: michaelr@outlook.com
Balance: 300.00 $
Mobile: 01009700003
Opened: November 2008
-----
Account Number : 9700000000
Name: Michael Jones
E-mail: m.jones@gmail.com
Balance: 1000.00 $
Mobile: 01009700000
Opened: December 2009
-----
Account Number : 9700000003
Name: John Roberto
E-mail: j.roberto@outlook.com
Balance: 90.00 $
Mobile: 01009700001
Opened: December 2008
-----
Account Number : 9700000009
Name: James Evans
E-mail: j.evans@gmail.com
Balance: 200.00 $
Mobile: 01009700009
Opened: May 2017
```


User_Manual

- Option 11: Quit
Choose this option to exit the program.

Thank you for choosing our services

Note:

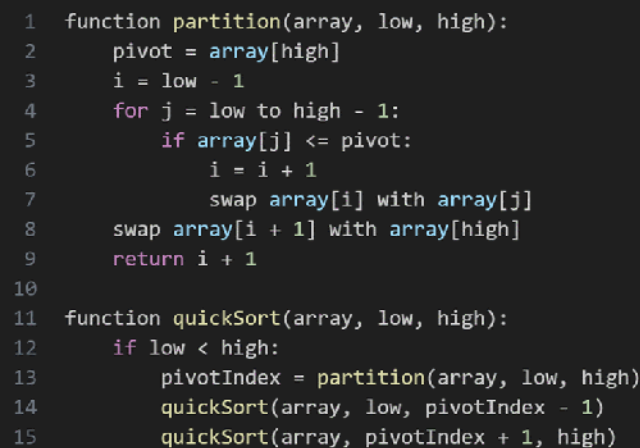
Throughout the program, please carefully follow the on-screen instructions and input valid data as required. If you encounter any errors or need assistance, refer to the program's instructions and error messages for guidance.

We hope this user manual provides clear instructions for navigating the Bank Account Management System. If you have any further questions or require assistance, please refer to the program's documentation or contact our support team for help.

Algorithms

Pseudo Code

Quick Sort:



```
1  function partition(array, low, high):
2      pivot = array[high]
3      i = low - 1
4      for j = low to high - 1:
5          if array[j] <= pivot:
6              i = i + 1
7              swap array[i] with array[j]
8      swap array[i + 1] with array[high]
9      return i + 1
10
11 function quickSort(array, low, high):
12     if low < high:
13         pivotIndex = partition(array, low, high)
14         quickSort(array, low, pivotIndex - 1)
15         quickSort(array, pivotIndex + 1, high)
```

In this pseudo code, the quickSort function takes an array, a low index, and a high index as input. It first checks if the low index is less than the high index, and if so, it finds the pivot index using the partition function and recursively calls itself on the subarrays to the left and right of the pivot.

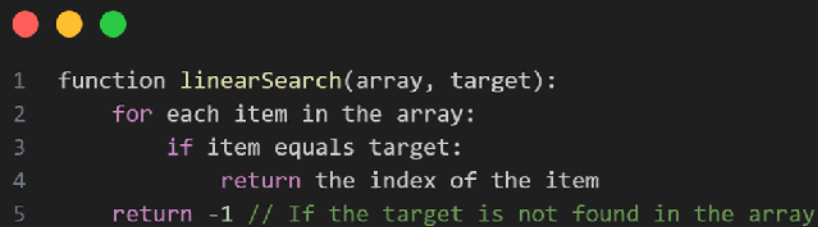
The partition function takes an array, a low index, and a high index as input. It selects the pivot element (in this case, the element at the high index), and then rearranges the elements in the array so that all elements less than or equal to the pivot are to the left of it, and all elements greater than the pivot are to the right of it. Finally, it returns the index of the pivot element after the rearrangement.

This pseudo code represents the basic idea of the quicksort algorithm, which is a popular and efficient sorting algorithm used in practice.

Algorithms

Pseudo Code

Linear Search:



```
1 function linearSearch(array, target):  
2     for each item in the array:  
3         if item equals target:  
4             return the index of the item  
5     return -1 // If the target is not found in the array
```

In this pseudo code, the linearSearch function takes an array and a target value as input. It then iterates through each item in the array, comparing it with the target value. If a match is found, the function returns the index of the item. If the entire array is traversed and no match is found, the function returns -1 to indicate that the target value is not present in the array.