**PROBLEM STATEMENT**

The task is to design and implement a basic bank management system using C programming. The system should allow the user to perform essential banking operations, such as adding new accounts, deleting accounts, and updating account details, with an emphasis on using structured programming principles. The program should incorporate algorithms and flowcharts to demonstrate the logic behind its operations and showcase how various constructs like variables, constants, operators, expressions, and conditional statements can be effectively used in C programming.

**INTRODUCTION**

The objective is to develop a basic bank management system using C programming. The program will simulate real-world banking operations, allowing users to add, delete, and update account details in an efficient and structured manner.

The system will be built using essential programming concepts such as variables, constants, operators, expressions, and conditional statements. These elements will work together to create an interactive environment where the user can manage banking transactions. Additionally, algorithms and flowcharts will be used to define the logic behind each operation, ensuring a well-organized approach to the design and functionality of the program.

By creating this system, we aim to practice the application of basic C programming constructs while enhancing our problem-solving and logical thinking skills. The assignment also focuses on the importance of writing clean, readable, and maintainable code. Through the use of conditional statements and efficient data handling techniques, the program will demonstrate how to implement and optimize common tasks such as account management in a simulated banking environment.

Ultimately, this assignment serves as a practical exercise to better understand and implement the core principles of C programming, while simulating a useful real-world application.

**SYSTEM DESIGN**

**Pseudo code :**

**START**

**// Define constants and structures**

**MAX\_ACCOUNTS = 100 // Maximum number of accounts**

**accounts = Array of Account[100] // Array to store accounts**

**accountCount = 0 // Counter for the number of accounts**

**// Define Account structure with fields:**

**Account:**

**accountNumber**

**name**

**balance**

**aadhaar**

**// Function prototypes (declaration):**

**addAccount()**

**viewAccount()**

**updateAccount()**

**deleteAccount()**

**deposit()**

**withdraw()**

**findAccountIndex(accountNumber)**

**validateAadhaar(aadhaar)**

**MAIN:**

**WHILE True:**

**DISPLAY Bank Management System menu:**

**1. Add Account**

**2. View Account**

**3. Update Account**

**4. Delete Account**

**5. Deposit**

**6. Withdraw**

**7. Exit**

**Get user input for choice**

**SWITCH choice:**

**CASE 1:**

**CALL addAccount()**

**CASE 2:**

**CALL viewAccount()**

**CASE 3:**

**CALL updateAccount()**

**CASE 4:**

**CALL deleteAccount()**

**CASE 5:**

**CALL deposit()**

**CASE 6:**

**CALL withdraw()**

**CASE 7:**

**PRINT "Exiting the system..."**

**EXIT PROGRAM**

**DEFAULT:**

**PRINT "Invalid choice! Please try again."**

**END**

**// Function to add a new account**

**addAccount():**

**Prompt user for account number**

**IF account number exists (CALL findAccountIndex(accountNumber)):**

**PRINT "Account with this number already exists!"**

**RETURN**

**Prompt user for Aadhaar number**

**IF Aadhaar number is invalid (CALL validateAadhaar(aadhaar)):**

**PRINT "Invalid Aadhaar number. It should be exactly 11 digits."**

**RETURN**

**Prompt user for account holder name**

**Prompt user for initial deposit amount**

**CREATE a new Account with the provided details**

**ADD the new account to the accounts array**

**INCREMENT accountCount by 1**

**PRINT "Account created successfully!"**

**// Function to view an account**

**viewAccount():**

**Prompt user for account number**

**FIND the index of the account using findAccountIndex(accountNumber)**

**IF account is found:**

**PRINT account details (account number, holder's name, Aadhaar number, balance)**

**ELSE:**

**PRINT "Account not found!"**

**// Function to update an account**

**updateAccount():**

**Prompt user for account number**

**FIND the index of the account using findAccountIndex(accountNumber)**

**IF account is found:**

**Prompt user for new name**

**Prompt user for new balance**

**UPDATE account details**

**PRINT "Account updated successfully!"**

**ELSE:**

**PRINT "Account not found!"**

**// Function to delete an account**

**deleteAccount():**

**Prompt user for account number**

**FIND the index of the account using findAccountIndex(accountNumber)**

**IF account is found:**

**SHIFT remaining accounts to remove the account from the array**

**DECREMENT accountCount by 1**

**PRINT "Account deleted successfully!"**

**ELSE:**

**PRINT "Account not found!"**

**// Function to deposit money into an account**

**deposit():**

**Prompt user for account number**

**FIND the index of the account using findAccountIndex(accountNumber)**

**IF account is found:**

**Prompt user for deposit amount**

**IF amount is positive:**

**ADD deposit amount to account balance**

**PRINT "Deposited [amount] successfully!"**

**ELSE:**

**PRINT "Invalid deposit amount!"**

**ELSE:**

**PRINT "Account not found!"**

**// Function to withdraw money from an account**

**withdraw():**

**Prompt user for account number**

**FIND the index of the account using findAccountIndex(accountNumber)**

**IF account is found:**

**Prompt user for withdrawal amount**

**IF amount is positive AND sufficient balance is available:**

**SUBTRACT withdrawal amount from account balance**

**PRINT "Withdrew [amount] successfully!"**

**ELSE:**

**PRINT "Insufficient funds or invalid amount!"**

**ELSE:**

**PRINT "Account not found!"**

**// Function to find account index by account number**

**findAccountIndex(accountNumber):**

**FOR each account in accounts array:**

**IF accountNumber matches the current account's account number:**

**RETURN the index of the account**

**RETURN -1 // Account not found**

**// Function to validate Aadhaar number (checks if it's exactly 11 digits)**

**validateAadhaar(aadhaar):**

**IF length of aadhaar is NOT 11:**

**RETURN -1 // Invalid Aadhaar number**

**FOR each character in aadhaar:**

**IF character is NOT a digit:**

**RETURN -1 // Invalid Aadhaar number**

**RETURN 0 // Aadhaar number is valid**

**END**

**Algorithms**

**1. Main Menu Algorithm:**

This algorithm repeatedly displays the main menu and prompts the user to select an operation. The program keeps running until the user chooses to exit.

**Steps:**

1. Display the bank management system menu options.
2. Wait for user input to select an operation.
3. Based on the user choice: Call the corresponding function (e.g., addAccount(), viewAccount(), etc.).
4. If the user chooses "7", exit the program.
5. If an invalid choice is entered, prompt the user again.
6. Repeat the process until the user chooses to exit.

### ****2. Add Account Algorithm:****

This algorithm is used to add a new account to the system.

**Steps:**

1. Prompt the user to enter an account number.
2. Check if the account number already exists by calling the findAccountIndex() function.
   * If the account number already exists, print a message and return.
3. Prompt the user to enter the Aadhaar number.
4. Validate the Aadhaar number by calling the validateAadhaar() function.
   * If the Aadhaar number is invalid, print an error and return.
5. Prompt the user to enter the account holder's name and initial deposit amount.
6. Create a new account object with the provided details.
7. Store the new account in the accounts array.
8. Increment the accountCount by 1.
9. Print a success message.

### ****3. View Account Algorithm:****

This algorithm is used to display the details of an existing account.

**Steps:**

1. Prompt the user to enter the account number.
2. Call the findAccountIndex() function to find the index of the account in the accounts array.
   * If the account is found, display the account details (account number, holder's name, Aadhaar number, and balance).
   * If the account is not found, print an error message.

### ****4. Update Account Algorithm:****

This algorithm is used to update an existing account's information (name and balance).

**Steps:**

1. Prompt the user to enter the account number to update.
2. Call the findAccountIndex() function to find the index of the account.
   * If the account is found:
     + Prompt the user to enter a new name for the account holder.
     + Prompt the user to enter a new balance.
     + Update the account's details.
     + Print a success message.
   * If the account is not found, print an error message.

### ****5. Delete Account Algorithm:****

This algorithm is used to delete an account from the system.

**Steps:**

1. Prompt the user to enter the account number to delete.
2. Call the findAccountIndex() function to find the index of the account.
   * If the account is found:
     + Shift the remaining accounts in the array to fill the gap left by the deleted account.
     + Decrement the accountCount by 1.
     + Print a success message.
   * If the account is not found, print an error message.

### ****6. Deposit Algorithm:****

This algorithm is used to deposit money into an existing account.

**Steps:**

1. Prompt the user to enter the account number where the deposit will be made.
2. Call the findAccountIndex() function to find the index of the account.
   * If the account is found:
     + Prompt the user to enter the deposit amount.
     + If the deposit amount is valid (greater than zero), add the amount to the account's balance.
     + Print a success message.
   * If the account is not found, print an error message.

### ****7. Withdraw Algorithm:****

This algorithm is used to withdraw money from an existing account.

**Steps:**

1. Prompt the user to enter the account number from which the withdrawal will be made.
2. Call the findAccountIndex() function to find the index of the account.
   * If the account is found:
     + Prompt the user to enter the withdrawal amount.
     + Check if the withdrawal amount is valid (greater than zero) and the account has enough balance.
     + If the conditions are met, subtract the withdrawal amount from the account's balance.
     + Print a success message.
     + If the account does not have sufficient funds or the amount is invalid, print an error message.
   * If the account is not found, print an error message.

### ****8. Find Account Index Algorithm:****

This algorithm searches for an account in the accounts array by its account number.

**Steps:**

1. Iterate through the accounts array.
2. For each account, check if the accountNumber matches the input.
3. If a match is found, return the index of the account.
4. If no match is found after checking all accounts, return -1 to indicate the account does not exist.

### ****9. Validate Aadhaar Algorithm:****

This algorithm validates the Aadhaar number entered by the user.

**Steps:**

1. Check if the length of the Aadhaar number is exactly 11 digits.
2. If the length is not 11, return -1 to indicate invalid Aadhaar number.
3. Iterate through each character in the Aadhaar number.
   * If any character is not a digit, return -1.
4. If all characters are digits and the length is 11, return 0 to indicate a valid Aadhaar number.

### ****Flowchart Process Details****:

1. **Start**: The program starts and displays the main menu.
2. **Main Menu**: The program waits for user input on which operation to select.
   * Based on the selected option, the program proceeds to one of the following operations:
     + Add Account
     + View Account
     + Update Account
     + Delete Account
     + Deposit
     + Withdraw
     + Exit
3. **Add Account**:
   * **Account Exists?** Checks if the account number already exists.
   * **Aadhaar Validated?**: Ensures Aadhaar is valid (11 digits).
   * After validation, the account is added to the list.
4. **View Account**:
   * **Account Found:** Searches for the account number. If found, it displays account details.
5. **Update Account**:
   * **Account Found:** Checks if the account exists and then allows the user to update the account details (name and balance).
6. **Delete Account**:
   * **Account Found:** Checks if the account exists and then deletes the account by shifting the other accounts.
7. **Deposit/Withdraw**:
   * For both operations, the system first checks if the account exists, then validates the amount for deposit/withdrawal, ensuring sufficient funds in case of withdrawal.
8. **Exit**: The program ends when the user chooses the exit option.

**IMPLEMENTATION**

Use of Arrays to Store Account Data

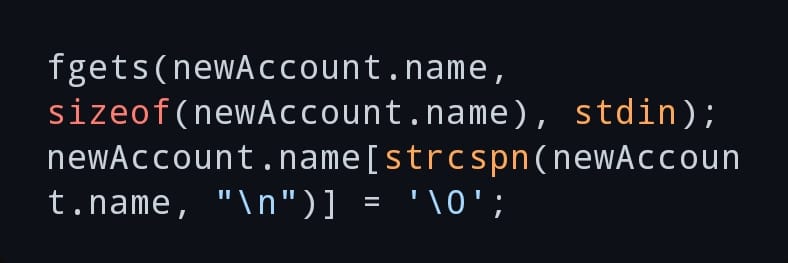
The system uses an array of structures to store account details (accountNumber, name, balance, aadhaar), with a predefined size of 100 accounts (MAX\_ACCOUNTS = 100).

Functions Implemented

1. addAccount: Adds a new account after validating account number uniqueness and Aadhaar format.
2. viewAccount: Displays account details by searching the account number.
3. updateAccount: Updates name and balance of an existing account.
4. deleteAccount: Removes an account by shifting elements in the array.
5. deposit: Adds a specified amount to the account balance after validation.
6. withdraw: Deducts an amount, ensuring sufficient funds.
7. findAccountIndex: Returns the index of an account or -1 if not found.
8. validateAadhaar: Checks if Aadhaar is numeric and of the correct length.

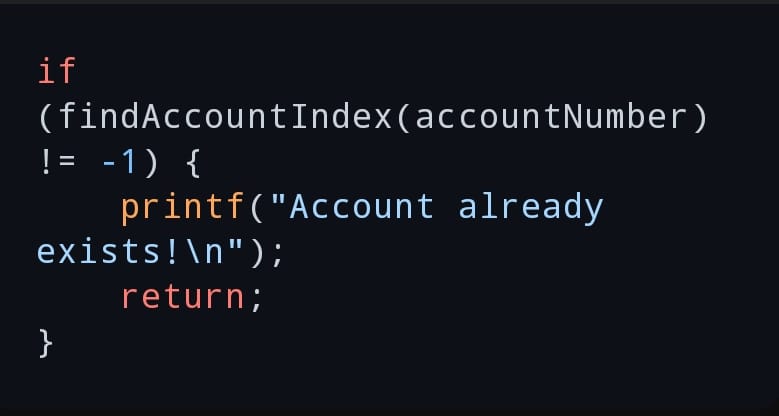
Key Code Snippets

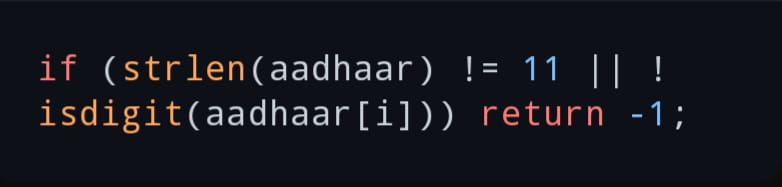
Handling User Input

****

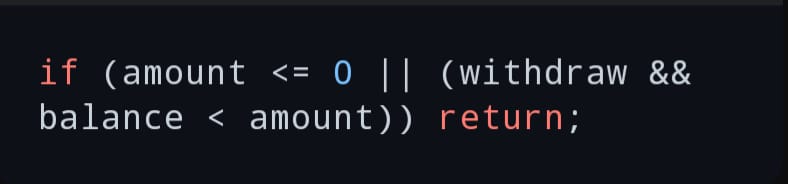
Choice Handling: A switch statement in the main menu directs user actions.

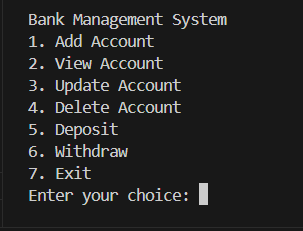
Error Handling

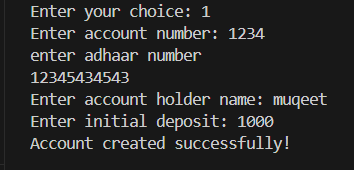
* ****Account Validation: Prevents duplicate account creation:
* Aadhaar Validation: Ensures Aadhaar is numeric and valid length:

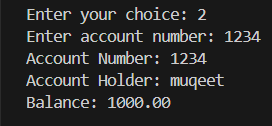
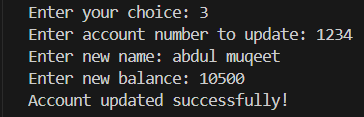
****

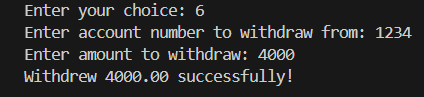
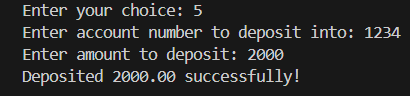
* Transaction Validation: Checks for positive amounts and sufficient balance:

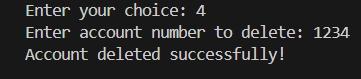


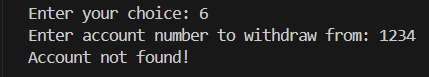
**Sample Input / Output**

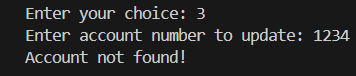
****

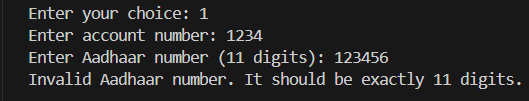
****

****

****

**Handling invalid inputs or error**

****

****

**TESTING AND VALIDATION**

How the Program Was Tested:

The program was tested for various banking operations like adding, viewing, updating, deleting accounts, and making deposits and withdrawals. It checks for valid and invalid inputs, ensuring proper handling of errors, like invalid Aadhaar numbers, insufficient funds, and duplicate account numbers.

summarizing test cases, inputs, expected outputs, and actual outputs.

|  |  |  |  |
| --- | --- | --- | --- |
| **SL No.** | **Test Case** | **Input** | **Expected Output** |
| **1)** | Add Account - Valid | Account: 12345, Aadhaar: 12345678901, Name: John, Deposit: 1000 | "Account created!" |
| **2)** | |  | | --- | | **Add Account - Duplicate** | | |  | | --- | | **Add Account - Duplicate** | | |  | | --- | | **Add Account - Duplicate** | |
| **3)** | |  | | --- | | **Add Account - Invalid Aadhaar** | | |  | | --- | | **Add Account - Invalid Aadhaar** | | |  | | --- | | **Add Account - Invalid Aadhaar** | |
| **4)** | |  | | --- | | **View Account - Valid** | | |  | | --- | | **View Account - Valid** | | |  | | --- | | **View Account - Valid** | |
| **5)** | |  | | --- | | **View Account - Invalid** | | |  | | --- | | **View Account - Invalid** | | |  | | --- | | **View Account - Invalid** | |
| **6)** | |  | | --- | | **Update Account** | | |  | | --- | | Account: 12345, Name: John Smith, Balance: 2000 | | |  | | --- | | "Account updated!" | |
| **7)** | Delete Account | Account: 12345 | "Account deleted!" |
| **8)** | **Deposit - Valid** | Account: 12345, Amount: 500 | "Deposited 500!" |
| **9)** | Withdraw - Valid | Account: 12345, Amount: 100 | "Withdrew 100!" |
| **10)** | **Withdraw - Insufficient Funds** | Account: 12345, Amount: 2000 | "Insufficient funds!" |

**CHALLENGES FACED**

1. Aadhaar Validation Issues

* Problem**:** The Aadhaar validation function checks for a strict length of 11 digits, but in reality, Aadhaar numbers are 12 digits long in India. Therefore, the program's validation is incorrect since it expects an 11-digit input but Aadhaar numbers are 12 digits long.
* Solution: Adjust the validation logic to check for exactly 12 digits, not 11.

**if (strlen(aadhaar) != 12) {**

**return -1; // Invalid Aadhaar number**

**}**

1. Input Handling

* Problem 1: The fgets function used for reading the account holder’s name can lead to an issue if the user inputs a string longer than the allocated space (50 characters). It could lead to buffer overflow if proper checks are not in place.
* Problem 2: After reading integers with scanf, there can be leftover newline characters (\n) in the input buffer. This can interfere with the subsequent input, such as when using fgets for the name input.
* Solution: Ensure that the input is properly sanitized and manage buffer overflow and unwanted characters.

**fgets(newAccount.name, sizeof(newAccount.name), stdin);**

**newAccount.name[strcspn(newAccount.name, "\n")] = '\0'; // Remove newline character**

**FUTURE SCOPE**

In my banking system project, I am looking to enhance its functionality and make it more user-friendly by incorporating several updates and improvements. These additions will focus on improving the overall experience for both users and administrators. I plan to implement features like:

1. Mobile and Online Banking
   * Enhanced mobile apps with AI-powered virtual assistants, biometric security, and real-time cross-border payments.
   * Integration with third-party services for managing multiple financial tasks from one platform.
2. Loan Management Systems
   * Automation of loan approvals using AI for faster, transparent processes.
   * Personalized loan offers based on customer data and credit behaviour.
   * Blockchain for secure, transparent loan disbursement.
3. Fraud Detection and Security
   * Use of AI and ML to detect fraud and secure transactions in real-time.
   * Expansion of biometric authentication methods and blockchain for enhanced security.
4. Investment Management
   * AI-driven robo-advisors for personalized investment recommendations.
   * Integration of cryptocurrencies and blockchain for diversified investment options.
   * AI for improved risk management in portfolios.

**CONCLUSION**

The project involved creating a Bank Management System using C programming, focusing on modular design and array-based implementation. The system performed banking operations like account creation, updates, deletion, deposits, and withdrawals. Key elements included structured pseudocode, validation mechanisms (e.g., Aadhaar verification), and error handling. Challenges addressed included input sanitization and Aadhaar validation logic.

Future Scope: Enhancements proposed included mobile banking, AI-driven fraud detection, and investment management tools.

Importance of Modular Programming and Arrays

Modular Programming:

* Improves code readability, reusability, and maintainability.
* Enables easier debugging and collaborative development.

Array-Based Implementation:

* Simplifies data storage and retrieval.
* Offers efficient handling of small to medium datasets.

Together, these approaches provided a structured, scalable system that could be expanded with advanced features.

**REFERENCE**

* <https://stackoverflow.com>
* <https://www.geeksforgeeks.org>
* <https://www.w3schools.com>