Bank Management System

1. Introduction:

The **Bank Management System** is a console-based program written in assembly language designed to simulate core banking functionalities. The project is intended as an educational tool for understanding low-level programming and how it applies to real-world scenarios like banking. By leveraging assembly language, this program illustrates the mechanics of secure account management, financial transactions, and user interaction through a menu-driven interface.

This project is a culmination of concepts such as data manipulation, secure input handling, and error management, all while adhering to best practices in assembly programming.

---

2. Code Structure Overview:

The program is logically divided into distinct sections to ensure modularity and maintainability:

Utilities -

- Helper functions and macros for repetitive tasks like screen clearing, new line creation, and string printing.
- These utilities provide a foundation for the core functionalities, reducing redundancy.

Menu System -

- A user-friendly interface that displays options and captures inputs to navigate the program.
- Designed for clarity and intuitive navigation.

Banking Operations -

- The heart of the program, implementing functionalities such as account creation, deposits, withdrawals, and more.
- Each operation is encapsulated in a separate procedure for modularity.

Security Features -

- PIN verification ensures that only authorized users can access sensitive operations.
- Dynamic PIN handling supports various PIN lengths for enhanced security.

---

3. Detailed Documentation:

Utilities -

The utility section consists of reusable procedures and macros. These components streamline development and ensure consistent behavior across the program.

1. **`clearScreen`**:
    - Purpose: Simulates a screen clear by printing multiple new lines to enhance visual clarity.
    - Usage: Called before displaying a menu or significant output.
2. **`newLine`**:
    - Purpose: Prints a newline to the console for better formatting.
    - Usage: Widely used for separating outputs or structuring menus.
3. **`printString`**:
    - Purpose: Macro for printing a null-terminated string.
    - Usage: Ensures consistent output for messages and prompts.
4. **`printNumber`**:
    - Purpose: Converts a numeric value into its ASCII equivalent and displays it.
    - Usage: Displays numeric data such as account balances.
5. **`checkAccountCreated`**:
    - Purpose: Verifies if an account exists before proceeding with an operation.
    - Usage: Acts as a safeguard against uninitialized account access.

Menu System -

The menu system provides a structured interface for navigating the program:

- **`DisplayMenu`**:
    - Prints the main menu with options for various banking operations.
    - Includes options like creating an account, withdrawing money, and exiting.
- **`GetInputMenuSystem`**:
    - Captures user input for menu navigation.

- Ensures input validation to prevent unexpected behavior.

Banking Operations -

The banking operations form the core of the program. Each operation is encapsulated in a dedicated procedure for clarity and reusability.

Option 1: Create Account (`op1`)

- **Description:**
  Allows the user to create a new account by entering an account name and PIN.
- **Features**:
  - Validates user input to ensure proper storage of account details.
  - Provides feedback upon successful account creation.

Option 2: Print Account Details (`op2`)

- **Description**:
  Displays account information, including the name, masked PIN, and current balance.
- **Features**:
  - Incorporates PIN verification to restrict unauthorized access.
  - Shows detailed account information in a formatted structure.
- **Scenario Example**:
  A user forgets their current balance and accesses this option to check the information after entering their PIN.

Option 3: Withdraw Money (`op3`)

- **Description**:
  Facilitates secure withdrawals with predefined denominations (e.g., Rs 1000, 2000, 5000, 10000).
- **Features**:
  - Checks if the account has sufficient funds before processing the transaction.
  - Displays error messages if the withdrawal amount exceeds the available balance.
- **Scenario Example**:
  A user attempts to withdraw Rs 5000 but only has Rs 3000 in the account. The system displays an error and prevents the withdrawal.

Option 4: Deposit Money (`op4`)

- **Description**:
  Enables users to deposit predefined amounts into their account.
- **Features**:
  - Updates the account balance dynamically.
  - Provides feedback on successful deposits.
- **Scenario Example**:
  A user deposits Rs 1000 and Rs 2000 in two separate transactions. The system accurately updates and reflects the new balance.

Option 5: Reset Account (`op5`)

- **Description**:
  Resets all account details, including the name, PIN, and balance.
- **Features**:
  - Verifies the user's PIN before performing the reset.
  - Ensures all account data is cleared securely.

Option 6: Modify Account Details (`op6`)

- **Description**:
  Allows users to update their account name and PIN.
- **Features**:
  - Overwrites previous details with the new inputs.
  - Incorporates safeguards to ensure data consistency.
- **Scenario Example**:
  A user wants to update their PIN from "1234" to "5678" for enhanced security.

Security Features -

The system implements robust security mechanisms:

- **PIN Verification**:
  - Ensures only authorized users can access sensitive operations.
  - Compares input PIN with the stored value, digit by digit.
- **Dynamic PIN Range**:
  - Supports flexible PIN lengths to accommodate varying security preferences.

Error Handling -

The program includes comprehensive error management to ensure smooth operation:

- Prevents users from performing actions on uninitialized accounts.
- Displays clear error messages for invalid inputs or insufficient balances.
- Allows users to retry operations or return to the main menu.

---

4. Constants and Data:

- **Account Details**:
  - `accountName`: Stores the name of the account holder.
  - `accountPIN`: Stores the account's PIN securely.
  - `totalAmount`: Maintains the current account balance.
- **Messages**: Predefined strings used for menus, prompts, and error messages.

5. Development Tools:

- **Assembler**: Emu 8086
- **Debugger**: Integrated debugging tools within emu 8086

---

6. Usage:

- Load the program in Emu 8086.
- Follow on-screen instructions to navigate through the menu.
- Perform operations such as creating accounts, deposits, and withdrawals.

7. OUTPUT: