Banking System Simulation Report

Overview:-

C programming:

Banking System Simulation: This program provides a simple menu-based interface to create accounts, deposit, withdraw, and check balances. I have coded the program in its simplest form, using functions for each attribute to work as the user chooses from the given options. The code will also throw an error when the user inputs the wrong command or an invalid input. The code methodology and working principles have been explained in detail with the following screenshots for the output and a Google Drive link that contains video as output.

1. Introduction to the Problem:

The problem is to simulate a basic banking system with features such as account creation, deposit, withdrawal, and balance check using the C programming language. The simulation should use structures and functions to manage accounts and their operations.

2. Solution Approach:

The solution involves using a menu-driven approach where users can choose different options to perform various banking operations. I've used structures to represent account information and functions to handle account-related operations.

3. Detailed Methodology with Working Code:

- After mentioning the required header files which are #include<stdio.h> and #include<stdlib.h> which are responsible for the code input/output and libraries for which we have used functions.

- I have used Structure which allows creating a compound data structure that can hold different types of variables under a single name. In the context of the banking system simulation, the 'struct' is used to define the structure of an "Account."

```
3
4   // Structure to represent an account
5   struct Account {
6       int accountNumber;
7       char accountHolder[50];
8       double balance;
9   };
10
```

The code needs to perform actions such as new account, deposit, withdraw amount, and balance inquiry. User may give any command from the mentioned action. In order to avoid

re-writing the logic again and again I implemented function call within the while loop in given conditions.

Function 'createAccount':-

This function is used to create a new account and store its details in the array of account structures.

```c
// Function to create a new account
void createAccount(struct Account *accounts, int *numAccounts) {
    printf("Enter account holder's name: ");
    scanf("%s", accounts[*numAccounts].accountHolder);

    accounts[*numAccounts].accountNumber = *numAccounts + 1;
    accounts[*numAccounts].balance = 0;

    (*numAccounts)++;
    printf("Account created successfully. Account number: %d\n", *numAccounts);
}
```

Arguments:

- **'struct Account *accounts':** A pointer to an array of 'struct Account' instances where account information will be stored.
- **'int *numAccounts':** A pointer to an integer indicating the total number of account. This will be updated after creating a new account.

Explanation:

1. The function prompts the user to enter the account holder's name using 'printf'.
2. It uses 'scanf' to read the account holder's name and store it in the 'accountHolder' member of the newly created account.
3. The account number is set to '*numAccounts + 1', ensuring the first account created gets an account number of 1.
4. The initial balance of the new account is set to 0.
5. '(*numAccounts)' is incremented by 1 to reflect the creation of a new account.
6. A Success message is printed with the new account number.

Function 'deposit':-

This function is used to deposit funds into an existing account.

```c
// Function to deposit amount into an account
void deposit(struct Account *accounts, int numAccounts, int accountNumber, double amount) {
    if (accountNumber >= 1 && accountNumber <= numAccounts) {
        accounts[accountNumber - 1].balance += amount;
        printf("Deposit successful. New balance: %.2f\n", accounts[accountNumber - 1].balance);
    } else {
        printf("Invalid account number.\n");
    }
}

```

Arguments:

- **'struct Account *accounts':** A pointer to an array of 'struct Account' instances.

- **‘int numAccounts’:** An integer indicating the total number of account.
- **‘int accountNumber’:** The account number to which the deposit should be made.
- **‘double amount’:** The amount to be deposited.

Explanation:

1. The function checks if the provided ‘accountNumber’ is within the valid range (1 to ‘numAccounts’).
2. If the ‘accounts[accountNumber -1].balance +=amount;’.
3. A success message is printer along with the updated balance.
4. If the account number is invalid, an error message is displayed.

Function ‘withdraw’:-

This function is used to withdraw funds from an existing account.

```
33  // Function to withdraw amount from an account
34  void withdraw(struct Account *accounts, int numAccounts, int accountNumber, double amount) {
35      if (accountNumber >= 1 && accountNumber <= numAccounts) {
36          if (accounts[accountNumber - 1].balance >= amount) {
37              accounts[accountNumber - 1].balance -= amount;
38              printf("Withdrawal successful. New balance: %.2f\n", accounts[accountNumber - 1].balance);
39          } else {
40              printf("Insufficient balance.\n");
41          }
42      } else {
43          printf("Invalid account number.\n");
44      }
45  }
```

Arguments:

- **‘struct Account *accounts’:** A pointer to an array of ‘struct Account’ instances.
- **‘int numAccounts’:** An integer indicating the total number of accounts.
- **‘int accountNumber’:** The account number from which the withdrawal should be made.
- **‘double amount’:** The amount to be withdrawn.

Explanation:

1. The function checks if the provided ‘accountNumber’ is within the valid range (1 to ‘numAccounts’).
2. If the account number is valid and the account balance is sufficient, the specified ‘amount’ is subtracted from the account’s balance.
3. A success message is printed along with the updated balance.
4. If the account number is invalid or the balance is insufficient, appropriate error messages are displayed.

Function ‘checkBalance’:-

This function is used to check the balance of an existing account.

```c
// Function to check account balance
void checkBalance(struct Account *accounts, int numAccounts, int accountNumber) {
    if (accountNumber >= 1 && accountNumber <= numAccounts) {
        printf("Account Holder: %s\n", accounts[accountNumber - 1].accountHolder);
        printf("Account Number: %d\n", accounts[accountNumber - 1].accountNumber);
        printf("Account Balance: %.2f\n", accounts[accountNumber - 1].balance);
    } else {
        printf("Invalid account number.\n");
    }
}
```

Arguments:

- **'struct Account *accounts':** A pointer to an array of 'struct Account' instances.
- **'int numAccounts':** An integer indicating the total number of accounts.
- **'int accountNumber':** The account number for which the balance should be checked.

Explanation:

1. The function checks if the provided 'accountNumber' is within the valid range (1 to 'numAccounts').
2. If the account number is valid, the account holder's name, account number, and balance are printed.
3. If the account number is invalid, an error message is displayed.

Function 'main':-

The 'main' function is the entry point of the program and contains the menu-driven logic for interacting with the banking system simulation. The 'main' function effectively manages the menu-driven interface, allowing users to interact with the banking system simulation by choosing various options. It orchestrates the execution of different functions based on the user's input.

```c
int main() {
    struct Account accounts[10]; // Maximum 10 accounts
    int numAccounts = 0;

    int choice, accountNumber;
    double amount;
```

- The 'main' function serves as the entry point of the program. It initializes an array of 'struct-Account' named 'accounts' to store account details and sets an initial value of 'numAccounts' to 0, indicating that there are no accounts at the beginning.
- Two integers ('choice, and 'accountNumber') and a double ('amount') are declared to store user input for menu choices and account operations.

```
65      while (1) {
66          printf("\nBanking System Menu:\n");
67          printf("1. Create Account\n");
68          printf("2. Deposit\n");
69          printf("3. Withdraw\n");
70          printf("4. Check Balance\n");
71          printf("5. Exit\n");
72          printf("Select option from above: ");
73          scanf("%d", &choice);
74
```

- The program enters a continuous loop using ‘while (1)’ to create an interactive menu driven interface for the banking system simulation.
- Within the loop, the program prints the main menu with various options for the user: create an account, deposit funds, withdraw funds, check balance, or exit the program.
- A ‘Switch’ statement is used to handle different menu choices based on the value of ‘choice’.

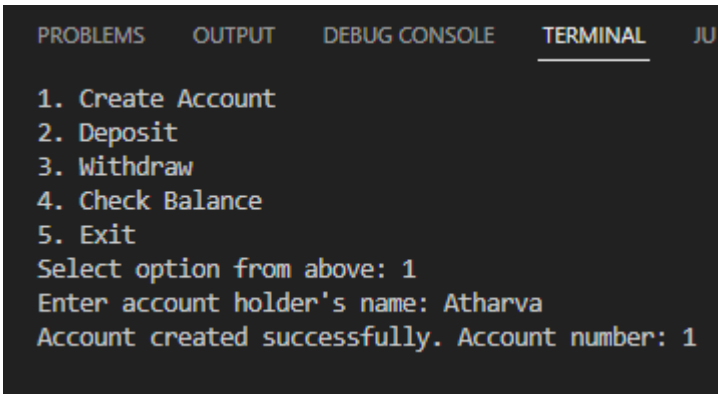
```
75          switch (choice) {
76              case 1:
77                  createAccount(accounts, &numAccounts);
78                  break;
79              case 2:
80                  printf("Enter account number: ");
81                  scanf("%d", &accountNumber);
82                  printf("Enter amount to deposit: ");
83                  scanf("%lf", &amount);
84                  deposit(accounts, numAccounts, accountNumber, amount);
85                  break;
86              case 3:
```

- A ‘switch’ statement is used to handle different menu choices based on the value of ‘choice’ the user make.
- There are certain case as required options, the cases contains the following functions which mentioned before, it takes the input from user and run the needed function which has been used.
- If the user enters an invalid choice, an error message is displayed. This is set to be as Default Case.
- The ‘return 0;’ statement indicates that the program executed successfully and is returning 0 to operating system.

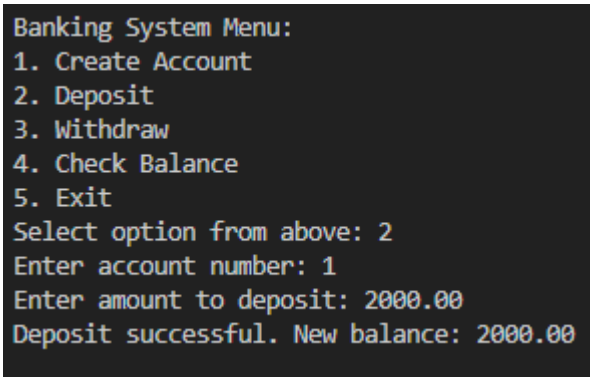
4. Outputs as Screenshots:

Given step by step output Screenshots:

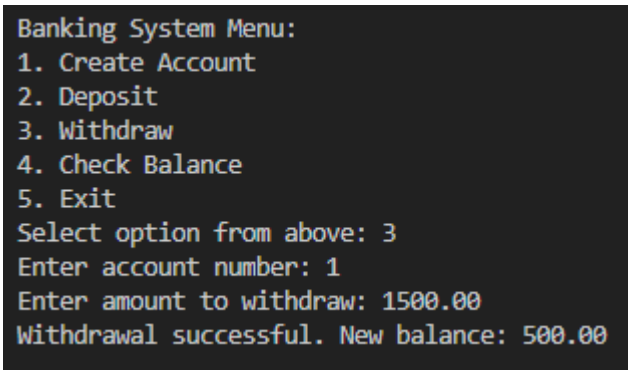
The window you will see on terminal when the code is initially run.

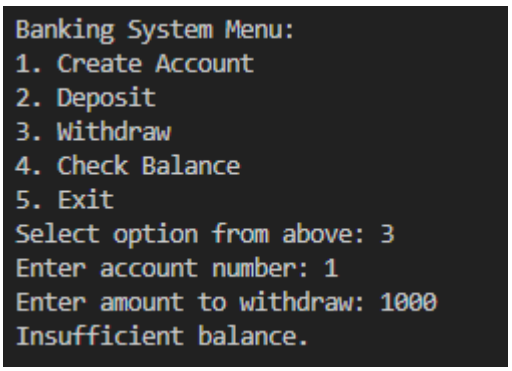


After successfully creating new account we move toward other tasks.



After creating and depositing we can now withdraw the amount, the code will also throw an error if the input balance is more than actual deposit.





The code will always throw an error whenever the user provides an invalid input.

```
Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Select option from above: 4
Enter account number: 1
Account Holder: Atharva
Account Number: 1
Account Balance: 500.00
```

The code will exit the loop after selecting the exit option.

```
Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Select option from above: 5
Exiting...
```

5. Outputs as Video

- The following Google Drive link contains video recording of program after running the code.

Google Drive link:
https://drive.google.com/drive/folders/1VETzSBzTWgNLATyTKkAftf1PLGFycIyV?usp=drive_link

6. Hardware Requirements:

The program is simple and doesn't have significant hardware requirements. It can run on most modern computers or virtual machines.

7. Software Requirements:

It needs a C compiler to compile and run the code. Here are the basic software requirements:

- C compiler (e.g., GCC)
- Text Editor or Integrated Development Environment (IDE) to write and edit the code (e.g., Visual Studio Code, etc)
- Terminal or Command Prompt to execute the compiled program