CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

C PROGRAMMING PROJECT

TOPIC : ATM TRANSACTION MANAGEMENT SYSTEM

Submitted by,

Ajin George Binu

Reg No: 2447103

Kevin Roy

Reg No: 2447127

Pranab Rai

Reg No: 2447137

Introduction

The Smart ATM System is a simulation of a basic banking system that enables users to manage their accounts through an ATM-like interface. This application allows users to perform essential banking operations, such as:

- Creating new accounts
- Depositing and withdrawing money
- Checking their balance
- Viewing a mini-statement of their most recent transactions
- Transferring funds between accounts

The application stores account details and transaction histories persistently in binary files, ensuring that data remains intact across sessions. Each user's account is protected by a PIN for security, while transactions such as deposits, withdrawals, and fund transfers are logged for transparency.

This system is implemented in C and leverages file handling for persistent storage of account information (accounts.bin) and transaction histories (transactions.bin). It supports up to 100 accounts and is designed with simplicity and ease of use in mind, providing an intuitive, menu-based interaction for users.

Functionalities

1. **Create New Account**
    - Allows the user to create a new account by entering their name and setting a PIN.
    - Each account is assigned a unique account number and starts with a balance of $0.
    - Account details (name, PIN, balance, recent transactions) are saved to the binary file accounts.bin.

2. **Deposit Money**
    - Users can deposit money into their account after authenticating with their account number and PIN.
    - The deposit amount is added to the user's balance and recorded in the recent transactions.
    - The transaction is saved in the transactions.bin file.
3. **Withdraw Money**
    - Users can withdraw money from their account, provided they have sufficient balance.
    - After successful authentication with account number and PIN, the withdrawn amount is deducted from the balance.
    - The transaction is recorded and saved in the transactions.bin file.
4. **Check Balance**
    - Users can check their current balance after authenticating with their account number and PIN.
5. **Mini Statement**
    - Users can view their last five transactions after successful authentication.
    - Each transaction is displayed along with the amount and type (Deposit, Withdrawal, Transfer).
6. **Transfer Funds**
    - Users can transfer funds from their account to another account.
    - The sender is authenticated using both the account number and PIN, while the receiver is authenticated by the account number only (no PIN required for the receiver).
    - The transfer is recorded for both the sender and receiver and saved in the transactions.bin file.
7. **Exit**
    - Exits the program after ensuring all data is saved.

Code Breakdown

Structures

Account Structure

- Represents an account with the following fields:
  - accountNumber: Unique identifier for the account.
  - name: Account holder's name.
  - pin: A 4-digit PIN for account security.
  - balance: Current balance of the account.
  - lastTransactions[5]: Circular buffer holding the last five transactions.
  - transactionCount: Number of transactions performed.

Functions

1. **void createAccount(struct Account accounts[], int *totalAccounts)**
   - Creates a new account by asking for the user's name and a 4-digit PIN.
   - Initializes the account with a balance of $0 and stores the account in memory.
   - Saves the account to accounts.bin using the saveAccounts() function.
2. **void deposit(struct Account *account, double amount, FILE *file)**
   - Deposits money into the specified account, updates the balance, and records the transaction in transactions.bin.
3. **void withdraw(struct Account *account, double amount, FILE *file)**
   - Withdraws money from the account if there is sufficient balance, updates the balance, and records the transaction in transactions.bin.

4. **void checkBalance(struct Account *account)**
    - Displays the current balance of the account.
5. **void miniStatement(struct Account *account)**
    - Prints the last five transactions from the account's lastTransactions array.
6. **int authenticate(struct Account accounts[], int totalAccounts, int accountNumber, int pin, int requirePin)**
    - Authenticates the user by checking their account number and PIN.
    - If requirePin is 1, both the account number and PIN are checked; otherwise, only the account number is validated (used in transfers).
7. **void transferFunds(struct Account *sender, struct Account *receiver, double amount, FILE *file)**
    - Transfers money from the sender to the receiver.
    - Both the sender's and receiver's balances are updated, and the transaction is recorded for both parties.
8. **void saveTransaction(FILE *file, struct Account *account, double amount, char *type)**
    - Saves a transaction (deposit, withdrawal, transfer) to transactions.bin in a binary format.
9. **void saveAccounts(struct Account accounts[], int totalAccounts)**
    - Saves the list of accounts to accounts.bin so that the account details persist after the program closes.
10. **void loadAccounts(struct Account accounts[], int *totalAccounts)**
    - Loads the list of accounts from accounts.bin at the start of the program to allow data persistence.

Files

1. **accounts.bin**
    - Stores all the account information in binary format.
    - Loaded when the program starts to retrieve account details and saved after any updates (e.g., deposits, withdrawals, new accounts).
2. **transactions.bin**
    - Stores transaction logs in binary format, including the account number, transaction type, amount, and the updated balance.
    - Each transaction (deposit, withdrawal, transfer) is appended to this file.

Data Flow

1. **Creating an Account**:
    - The user provides their name and sets a PIN.
    - A new Account is created, initialized, and stored in the accounts array.
    - The accounts array is saved to accounts.bin for persistence.
2. **Performing Transactions (Deposit, Withdrawal, Transfer)**:
    - The user authenticates by entering their account number and PIN.
    - The transaction (deposit, withdrawal, or transfer) is processed and reflected in the account's balance.
    - The transaction is recorded in transactions.bin and the updated account details are saved to accounts.bin.
3. **Loading Data**:
    - At the start of the program, accounts are loaded from accounts.bin using loadAccounts(), allowing users to access their saved accounts.

Error Handling

- Invalid inputs, such as incorrect PIN or insufficient funds, are handled with appropriate error messages.
- When creating an account, the system ensures that the account limit is not exceeded.
- During transfers, only the sender is authenticated using their PIN, and the receiver is validated by account number.

Source Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_ACCOUNTS 100

// Define a structure for user accounts
struct Account {
    int accountNumber;
    char name[50];
    int pin;
    double balance;
    double lastTransactions[5];
    int transactionCount;
};

// Define a structure for transactions (to be saved in the binary file)
struct Transaction {
    int accountNumber;
    char type[20];
    double amount;
    double newBalance;
```

```
};

// Function prototypes
void createAccount(struct Account accounts[], int *totalAccounts);
void deposit(struct Account accounts[], int index, double amount, FILE *file);
void withdraw(struct Account accounts[], int index, double amount, FILE *file);
void checkBalance(struct Account accounts[], int index);
void miniStatement(struct Account accounts[], int index);
int authenticate(struct Account accounts[], int totalAccounts, int accountNumber, int pin);
void transferFunds(struct Account accounts[], int senderIndex, int receiverIndex, double amount, FILE *file);
void saveTransaction(FILE *file, struct Account accounts[], int index, double amount, const char *type);
void saveAccounts(struct Account accounts[], int totalAccounts);
void loadAccounts(struct Account accounts[], int *totalAccounts);

int main() {
    struct Account accounts[MAX_ACCOUNTS];
    int totalAccounts = 0;
    int accountNumber, pin, authenticatedAccountIndex;
    int choice;
    double amount;

    // Load accounts from binary file
    loadAccounts(accounts, &totalAccounts);

    FILE *transactionFile = fopen("transactions.bin", "ab+");
    if (transactionFile == NULL) {
        printf("Error opening transaction file.\n");
        return 1;
    }
```

```
    do {
        // Main menu
        printf("\n--- Smart ATM System ---\n");
        printf("1. Create New Account\n");
        printf("2. Deposit Money\n");
        printf("3. Withdraw Money\n");
        printf("4. Check Balance\n");
        printf("5. Mini Statement\n");
        printf("6. Transfer Funds\n");
        printf("7. Exit\n");
        printf("Choose an option: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createAccount(accounts, &totalAccounts);
              saveAccounts(accounts, totalAccounts); // Save accounts
after creation
                break;
            case 2:
                printf("Enter Account Number: ");
                scanf("%d", &accountNumber);
                printf("Enter PIN: ");
                scanf("%d", &pin);
                  authenticatedAccountIndex = authenticate(accounts,
totalAccounts, accountNumber, pin);
                if (authenticatedAccountIndex != -1) {
                    printf("Enter amount to deposit: ");
                    scanf("%lf", &amount);
                        deposit(accounts, authenticatedAccountIndex,
amount, transactionFile);
                     saveAccounts(accounts, totalAccounts); // Save
accounts after deposit
```

```
            } else {
                printf("Invalid Account Number or PIN.\n");
            }
            break;
        case 3:
            printf("Enter Account Number: ");
            scanf("%d", &accountNumber);
            printf("Enter PIN: ");
            scanf("%d", &pin);
              authenticatedAccountIndex = authenticate(accounts,
totalAccounts, accountNumber, pin);
            if (authenticatedAccountIndex != -1) {
                printf("Enter amount to withdraw: ");
                scanf("%lf", &amount);
                   withdraw(accounts, authenticatedAccountIndex,
amount, transactionFile);
                  saveAccounts(accounts, totalAccounts); // Save
accounts after withdrawal
            } else {
                printf("Invalid Account Number or PIN.\n");
            }
            break;
        case 4:
            printf("Enter Account Number: ");
            scanf("%d", &accountNumber);
            printf("Enter PIN: ");
            scanf("%d", &pin);
              authenticatedAccountIndex = authenticate(accounts,
totalAccounts, accountNumber, pin);
            if (authenticatedAccountIndex != -1) {
               checkBalance(accounts, authenticatedAccountIndex);
            } else {
                printf("Invalid Account Number or PIN.\n");
            }
```

```
                break;
            case 5:
                printf("Enter Account Number: ");
                scanf("%d", &accountNumber);
                printf("Enter PIN: ");
                scanf("%d", &pin);
                  authenticatedAccountIndex = authenticate(accounts,
totalAccounts, accountNumber, pin);
                if (authenticatedAccountIndex != -1) {
                 miniStatement(accounts, authenticatedAccountIndex);
                } else {
                    printf("Invalid Account Number or PIN.\n");
                }
                break;
            case 6:
                printf("Enter Account Number: ");
                scanf("%d", &accountNumber);
                printf("Enter PIN: ");
                scanf("%d", &pin);
                  authenticatedAccountIndex = authenticate(accounts,
totalAccounts, accountNumber, pin);
                if (authenticatedAccountIndex != -1) {
                    int transferAccountNumber;
                    printf("Enter account number to transfer to: ");
                    scanf("%d", &transferAccountNumber);
                    int transferAccountIndex = -1;
                    for (int i = 0; i < totalAccounts; i++) {
                                if  (accounts[i].accountNumber  ==
transferAccountNumber) {
                            transferAccountIndex = i;
                            break;
                        }
                    }
                    if (transferAccountIndex != -1) {
```

```
                        printf("Enter amount to transfer: ");
                        scanf("%lf", &amount);
                                        transferFunds(accounts,
authenticatedAccountIndex,      transferAccountIndex,      amount,
transactionFile);
                      saveAccounts(accounts, totalAccounts); // Save
accounts after transfer
                    } else {
                        printf("Transfer account not found.\n");
                    }
                } else {
                    printf("Invalid Account Number or PIN.\n");
                }
                break;
            case 7:
               printf("Thank you for using the Smart ATM System.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 7);

    fclose(transactionFile);
    return 0;
}

// Function to create a new account
void createAccount(struct Account accounts[], int *totalAccounts) {
    if (*totalAccounts >= MAX_ACCOUNTS) {
            printf("Account limit reached. Cannot create more
accounts.\n");
        return;
    }
    struct Account newAccount;
```

```
    newAccount.accountNumber = 100 + *totalAccounts + 1;
    printf("Enter account holder's name: ");
    scanf(" %[^\n]", newAccount.name);
    printf("Set a 4-digit PIN: ");
    scanf("%d", &newAccount.pin);
    newAccount.balance = 0;
    newAccount.transactionCount = 0;

    accounts[*totalAccounts] = newAccount;
    (*totalAccounts)++;

     printf("Account created successfully. Account Number: %d\n",
newAccount.accountNumber);
}

// Function to deposit money
void deposit(struct Account accounts[], int index, double amount, FILE
*file) {
    if (amount <= 0) {
        printf("Invalid amount.\n");
        return;
    }
    accounts[index].balance += amount;
    accounts[index].lastTransactions[accounts[index].transactionCoun
t % 5] = amount;
    accounts[index].transactionCount++;
         printf("Deposit successful. New balance: $%.2f\n",
accounts[index].balance);

    saveTransaction(file, accounts, index, amount, "Deposit");
}

// Function to withdraw money
```

```
void withdraw(struct Account accounts[], int index, double amount, FILE *file) {
    if (amount <= 0) {
        printf("Invalid amount.\n");
        return;
    }
    if (amount > accounts[index].balance) {
        printf("Insufficient balance.\n");
        return;
    }
    accounts[index].balance -= amount;
    accounts[index].lastTransactions[accounts[index].transactionCount % 5] = -amount;
    accounts[index].transactionCount++;
        printf("Withdrawal successful. New balance: $%.2f\n", accounts[index].balance);

    saveTransaction(file, accounts, index, -amount, "Withdrawal");
}

// Function to check balance
void checkBalance(struct Account accounts[], int index) {
    printf("Current balance: $%.2f\n", accounts[index].balance);
}

// Function to print mini statement
void miniStatement(struct Account accounts[], int index) {
    printf("Mini Statement:\n");
        int start = accounts[index].transactionCount >= 5 ? accounts[index].transactionCount - 5 : 0;
    for (int i = start; i < accounts[index].transactionCount; i++) {
                printf("Transaction %d: %+.2f\n", i + 1, accounts[index].lastTransactions[i % 5]);
    }
```

```
}

// Function to authenticate user
int authenticate(struct Account accounts[], int totalAccounts, int accountNumber, int pin) {
    for (int i = 0; i < totalAccounts; i++) {
            if (accounts[i].accountNumber == accountNumber && accounts[i].pin == pin) {
            return i;
        }
    }
    return -1;
}

// Function to transfer funds
void transferFunds(struct Account accounts[], int senderIndex, int receiverIndex, double amount, FILE *file) {
    if (amount <= 0) {
        printf("Invalid amount.\n");
        return;
    }
    if (amount > accounts[senderIndex].balance) {
        printf("Insufficient balance for transfer.\n");
        return;
    }

    accounts[senderIndex].balance -= amount;
    accounts[receiverIndex].balance += amount;

    accounts[senderIndex].lastTransactions[accounts[senderIndex].transactionCount % 5] = -amount;
    accounts[senderIndex].transactionCount++;
```

```
    accounts[receiverIndex].lastTransactions[accounts[receiverIndex]
.transactionCount % 5] = amount;
    accounts[receiverIndex].transactionCount++;

        printf("Transfer successful. New balance: $%.2f\n",
accounts[senderIndex].balance);

    saveTransaction(file, accounts, senderIndex, -amount, "Transfer
Sent");
    saveTransaction(file, accounts, receiverIndex, amount, "Transfer
Received");
}

// Function to save accounts to a binary file
void saveAccounts(struct Account accounts[], int totalAccounts) {
    FILE *file = fopen("accounts.bin", "wb");
    if (file == NULL) {
        printf("Error saving accounts.\n");
        return;
    }
    fwrite(accounts, sizeof(struct Account), totalAccounts, file);
    fclose(file);
}

// Function to load accounts from a binary file
void loadAccounts(struct Account accounts[], int *totalAccounts) {
    FILE *file = fopen("accounts.bin", "rb");
    if (file != NULL) {
          *totalAccounts = fread(accounts, sizeof(struct Account),
MAX_ACCOUNTS, file);
        fclose(file);
    } else {
        printf("No existing account data found. Starting fresh.\n");
    }
```

```
}

// Function to save a transaction to a binary file
void saveTransaction(FILE *file, struct Account accounts[], int index,
double amount, const char *type) {
    struct Transaction transaction;
    transaction.accountNumber = accounts[index].accountNumber;
    strcpy(transaction.type, type);
    transaction.amount = amount;
    transaction.newBalance = accounts[index].balance;

    fwrite(&transaction, sizeof(struct Transaction), 1, file);
}
```

Output Screenshots

```
No existing account data found. Starting fresh.

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 1
Enter account holder's name: Ajin
Set a 4-digit PIN: 1234
Account created successfully. Account Number: 101

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 1
Enter account holder's name: Kevin
Set a 4-digit PIN: 1235
Account created successfully. Account Number: 102

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 1
Enter account holder's name: Pranab
Set a 4-digit PIN: 1236
Account created successfully. Account Number: 103
```

```
--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 106
Enter PIN: 1238
Invalid Account Number or PIN.

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 101
Enter PIN: 1234
Enter amount to deposit: 2000
Deposit successful. New balance: $2000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 102
Invalid choice. Please try again.
```

```
--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 102
Enter PIN: 1235
Enter amount to deposit: -4000
Invalid amount.

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 102
Enter PIN: 12356
Invalid Account Number or PIN.

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 102
Enter PIN: 1235
Enter amount to deposit: 4000
Deposit successful. New balance: $4000.00
```

```
--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 2
Enter Account Number: 103
Enter PIN: 1236
Enter amount to deposit: 6000
Deposit successful. New balance: $6000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 4
Enter Account Number: 101
Enter PIN: 1234
Current balance: $2000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 4
Enter Account Number: 102
Enter PIN: 1235
Current balance: $4000.00
```

```
--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 4
Enter Account Number: 103
Enter PIN: 1236
Current balance: $6000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 6
Enter Account Number: 103
Enter PIN: 1236
Enter account number to transfer to: 101
Enter amount to transfer: 2000
Transfer successful. New balance: $4000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 4
Enter Account Number: 101
Enter PIN: 1234
Current balance: $4000.00
```

```
--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 5
Enter Account Number: 103
Enter PIN: 1236
Mini Statement:
Transaction 1: +6000.00
Transaction 2: -2000.00

--- Smart ATM System ---
1. Create New Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Mini Statement
6. Transfer Funds
7. Exit
Choose an option: 7
```