

Project Report

Name : Sahil Dabra

Sap Id. : 590023527

Batch : 60

Abstract

The main intention of this project is to create a simple ATM Simulator using C language.

ATM is something we interact with daily for our banking needs. An ATM allows a person to withdraw cash, check balance, and also change PIN without standing in the queue of a bank.

So this project simulates all such operations. It also implements a file handling concept through which the user credentials (PIN, Account Balance) will be saved after you exit the program.

The program is menu-driven where a user can select certain options to carry out some banking operations. This project would be the simplest version to understand how real banking software would work with file handling and a secure establishment.

Problem Definition

What's the problem? Manual banking is slow and requires human interaction for every small transaction. We need a computerized system that can handle basic banking needs automatically and securely.

What's the Solution? I will write a C program that essentially needs to work like an ATM and therefore, it will:

1. Need a user PIN to access the main menu.
2. Denies access/locks the account after three attempts of entering the incorrect PIN (Security).
3. Have a menu to choose from (Check Balance, Withdraw Cash, Deposit Cash, Change PIN).
4. Have a mini statement of the last 5 transactions.
5. Need to write/read from files so that everything is in permanentized format and so that every time you come back to the program, it will be on record and updated.

System Design

1 Flowchart Overview

The system works in a linear flow:

1. **Start** -> Load Data from File.
2. Login Loop: Ask for PIN.
 - o If Correct -> Go to Main Menu.
 - o If Wrong -> Increment counter. If counter reaches 3, Lock Account and Exit.
3. Main Menu:
 - o Option 1: Check Balance -> Show bal.

- o Option 2: Withdraw - Check if bal > amount. If yes, deduct and save.
 - o Option 3: Deposit - Add to bal and save.
 - o Option 4: Mini Statement - Read transactions.txt.
 - o Option 5: Change PIN - Update pin variable and save.
 - o Option 6: Exit.
4. End.

2 Algorithm (Main Logic)

1. Initialize global variables for PIN, Lock status, and Balance.
2. Call load() function to read account.txt.
3. Check if lock is 1. If yes, print error and stop.
4. Loop 3 times asking for PIN.
5. If PIN matches, break loop. If loop finishes without match, set lock=1, call save(), and exit.
6. Start an infinite while(1) loop for the menu.
7. Take user choice (ch).
8. Use if-else if ladder to execute specific functions based on choice.
9. Update files immediately after any change (Deposit/Withdraw/PIN Change).

Implementation Details

The project is implemented in C Language. I used standard libraries like stdio.h for input/output and string.h for string operations. The core concept used here is File Handling.

Key Functions Used:

- load() & save(): These functions are responsible for reading the current balance/PIN from the hard drive and saving it back when it changes.
- void save() {
 FILE *fp = fopen("account.txt", "w");
 fprintf(fp, "%d %d %f", pin, lock, bal);
 fclose(fp);
}
- addTrans(): This tracks the history. Every time money is moved, this function adds a line to the transaction file.
- trimOld(): This is a logic I wrote to make sure the mini-statement doesn't get too big. It reads the whole file into an array and only rewrites the last 5 lines back to the file.
- chnagepin(): A simple function to update the security code.
- void chnagepin(): A simple function to change pin

Testing & Results

I have created a default account and tested the project with various scenarios to check if it handles data correctly.

Test Case 1: Successful Login

- **Input:** PIN = 1234 (Default)
- **Output:** "you are logged in" and Menu appears.

Test Case 2: Wrong PIN (Security)

- **Input:** Entered wrong PIN 3 times.
- **Output:** "account is locked".
- **Verification:** Restarted the app, and it said "acc is locked," proving the file save worked.

Test Case 3: Withdrawal

- **Initial Balance:** 5000
- **Input:** Withdraw 1000.
- **Output:** "Withdrawn." New Balance displayed: 4000.

Test Case 4: Insufficient Balance

- **Balance:** 4000
- **Input:** Withdraw 5000.
- **Output:** "balance is not enough".

Conclusion

This project successfully simulates the working of an ATM. It helped me understand how C handles files and how to create a menu-driven program. It handles basic errors (like low balance) and security (like account locking).

References

1. Youtube tutorials on C File I/O.
2. Class Notes (UPES, Semester 1).