Mini Project : Banking System

Objective:

Your project on Banking management System with options to create accouint , transactions , deposits , etc...

Program uses structure to store account holders details.

User Interface:

The application when executed, should display a menu as given below.

$./bank

----------MENU----------
c/C: Create account.
h/H: Transaction history (minimum --> last 5).
w/W: Withdraw amount.
d/D : Deposit amount.
b/B: Balance enquery.
t/T : Transfer money.
e/E: Display all accounts details.
s/S: Save the accounts info in file
f/F: Finding / searching for specific account.
q/Q: Quit from app

Requirements:

- Every new account should contain **Account_number , Account_name** , **Account_balance** ,**Account_Transactions ,Account_Transactions_count , contact number ,** if require , many more can be added.
- Here , **Account_Transactions** must be anotherr stucture to store transactions details like, TYPE of transaction (**withdraw / deposit**) , transaction_ID (**unique number --> UINT32_t**).
- Make sure that , duplicates accounts should not be present.
- Different accounts can create on same name but , account numbewr must be different.

DELIVERABLES:

1. This app should contain user-defined functions for each and every task.
   Ex: Create_account() , withdraw() , transfer() , etc....

2. Use makefile and make tool to manage the project.

3. Use **readme.txt** to explain the usage of the project, how to compile, execute etc..

4. If we **re-launch** the app , old / previows data should be available.

5. Deliver the project, in a folder(named your ID), containing all source files, headerfiles, makefile, and readme.txt.

Project Version1:

1. FileHandling : File based functions like save(), syncfile(), should use to store data in file.
2. For every function , separate file should be implemented.
3. Use structure pointer and implement by using SLL.

SUGGESTION:

A) Use typedef, enum, union where-ever applicable.
B) Use separate header file to keep all structure, union , and typedefs.