Sam Freed
Pavol Černý and Mohamed Salman
ECEN 1310
30 April 2019

ATM Project

This project most definitely challenged my programming skills and debugging abilities. I have had a lot of new interactions with balancing input and output, working within libraries and the creation of easily adaptable and editable programs, and understanding the interactions between storage and functions that access it.

First, my design choices. Using `printf` and a lot of newline `\n` characters, I created a text-based UI that displays various options and waits for input from the user to determine the next set of actions. Instead of having four to eight standard buttons as most actual ATMs do, numerical input allows for a more adaptable and general input structure and makes it very obvious as to which number leads to which next set of options. It also frequently utilizes `%s` and `%zu` modifiers in order to present the user with more specific details about the actions they have completed and their current status in the user system. The system frequently pauses after an action has been completed in order to prevent the user from being overwhelmed with the new menu each time and to, once again, allow for transparency and complete understanding of the user's interactions.

The users are stored in an array that is generated when the program is started, with a library-defined structure type `user`, which stores the user's inputted information including their legal name, address, age, and phone number. It also automatically begins the user's account with a balance of zero and a generated ten-digit phone number. The user also inputs a username and a numerical password for added privacy, preventing random users from easily accessing the personal information listed above. I could also have implemented this structure using a linked list setup; however, this would have caused difficulty when attempting to edit inputted data. Instead of simply accessing the array address and then overwriting with new data, the program would have to scan through each list address for the correct user and could easily have lost data if multiple operations were to have occurred at once. The array structure allows for a solid, stable location for all the data, keeping it all together in relatively close memory locations and expanding the array when it is needed to so as to save space as efficiently as possible.

In the future, this program could be made a lot more secure. Passwords are inputted and stored as plaintext, but (if it is within the capacity of C programming) the input from stdin could be hidden as many websites have implemented today. The passwords could also be stored using encryption, with algorithms based on other user-inputted data used to create a semi-random encryption pattern. This would make it at least a little harder to retrieve a user's password, especially relative to plaintext storage. The other way this could be improved would be exporting user data to another file, likely a database file, so that it would remain between runnings of the program and not require the admin to ask for users to input information again after the program starts. This database could eventually lead to a cloud-based system, with individual terminals running the ATM application at the same time and checking against a centralized, secure database, once again acting within the abilities of C.

The `bash` file included allows for easier repeated input and checking of ATM function. It uses a `.txt` file input in order to maintain the newline characters and their relations to `scanf`, especially since this program would be hard to develop using command line `argc/argv` values and even harder for users to understand. The `bash` file progresses as follows:

1. Signs into the admin account.
2. Creates a new user.
3. Signs out of admin and into the created user.
4. Runs the various functions of the user, including deposits, withdrawals, account information edits, and attempts to transfer cash between two accounts. After each function that edits user data, it prints the full user info.
5. Returns to the admin account and creates another user.
6. Transfers a balance from the first created account to the new account.
7. Deletes the second account.
8. Checks the second account login, which fails.
9. Clears the user array and successfully shuts down the application.

This script in itself could be improved by better showing the input sent to `stdin`, but due to limited knowledge of `bash` I was unable to show this effectively.