C Project Extension - Digital Receipt System

Group 17 : Wei Yi Tee, Soon Zhi Ho (Brandon), Zirun Zhai, Ivy Tam

September 27, 2020

1 Extension

1.1 Motivation and Use Of Extension

Paper receipts are an unexpectedly large source of pollution. The production of paper receipts emits 4 billion pounds of CO2, uses approximately 13.2 billion gallons of water and 12.4 million trees every year just in America [1]. Our group has decided to create a digital receipt system as an extension that would allow businesses to eliminate the need for paper receipts. This would mitigate the negative impact that paper receipts have on the environment. Moreover, digital receipts will allow for completely contactless payments, making social distancing due to the ongoing pandemic easier. These would allow our world to recover and improve.

For example, a restaurant could utilise our extension so that the cashier generates receipts for its customers according to their meals. Editable orders allow the cashier to amend any careless mistakes made in a hurry or add orders, e.g. if a customer decides to have dessert.

1.2 Design Details Of Extension

The finalised design (Figure 1b below) of our extension - a digital receipt system, first involves a login prompt for a merchant. The merchant enters an ID checked against existing merchant IDs. If the ID is found, the merchant will be prompted for their password. If the ID is not found, the merchant will be prompted for registration. Registering involves providing a password and a text file containing a menu of items. A folder corresponding to the merchant will be created which will store the menu and receipts issued by the merchant.

After successfully logging in, the email address and password of the merchant will be requested so that the system will be able to send receipts to customers after it has been authenticated. The merchant's menu will be automatically retrieved and loaded.

Once the merchant's data has been initialised, he/she can choose to carry out an action from a list, as shown below.

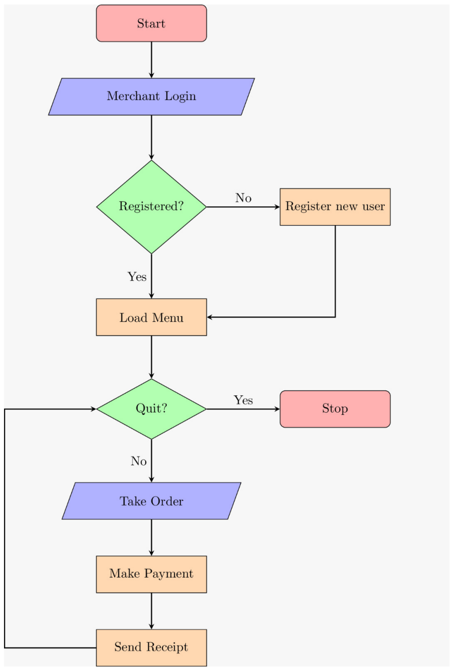
0) Quits the program

1) Take order: A new empty order is created and the menu is printed to display the list of items on sale. The merchant is responsible for adding items by specifying the item's ID and quantity. After finalising the order, the merchant can either choose to have the order paid immediately or store it in a list of unpaid orders to pay later. Paying immediately prompts the merchant for a payment method and follows with the same principle as 4).

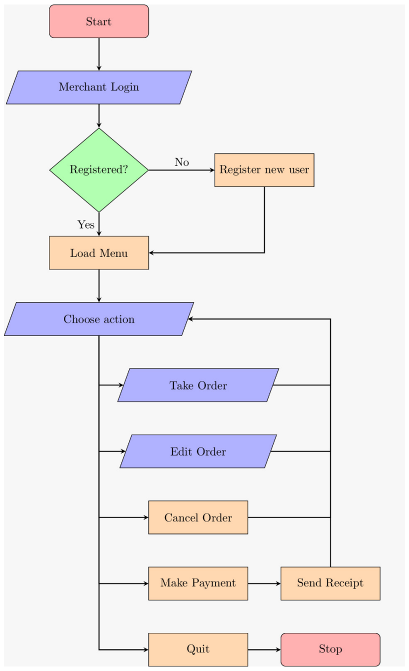
2) Edit order: An unpaid order is retrieved by entering the number corresponding to the customer name which can then be edited, and when finalised has the same options for payment as take order.

3) Cancel order: Using the same principle of retrieving for 2), an unpaid order can be removed.

4) Pay order: Using the same principle of retrieving for 2), an unpaid order will be made into a receipt containing the order and the total price to pay. The merchant is then prompted for a payment type which will be recorded in the receipt. It will be stored in the merchant's folder uniquely identified by the current time. The customer's email address will be requested to which the receipt will then be sent.



(a) Flow Chart Of Initial Design



(b) Flow Chart of Final Design

Figure 1: The final design was adapted to also suit businesses which have a pay later procedure (e.g. for restaurants) and allow for orders to be edited and cancelled.

1.3 Challenges in Extension Implementation

As in the initial design (Figure 1a), our program initially had a more sequential control flow which limited the choice of operations that can be done by the user. We decided to introduce a switch statement for actions and a list for storing orders.

Upon attempting to implement an automatic test suite, we realized that our functions needed to be able to interact with text files for comparison against expected behaviour. Hence, we had to modify functions involving I/O operations to be able to read from and write to both the standard I/O streams as well as text files.

We came up with the idea of emailing receipts to customers for their personal reference. After a bit of research, we realised that we needed to connect to a Simple Mail Transfer Protocol (SMTP) server to be able to send emails. We tried creating our own network sockets in C, but weren't able to initiate an encrypted session. Therefore, we decided to make use of the Python SMTPlib to write a short script for user authentication and sending emails. We then execute it from our main C program by the *system()* function.

For security considerations, we stored the passwords of registered users as hashes. Although peppers [2] are controversial in industrial practice due to the lack of a mechanism to rotate keys, we decided to attempt an implementation of a random pepper appended to the end of each user input password before hashing with the djb2 function which is both computed efficiently and distributed well. However, this meant that we had to use brute force to identify the pepper when authenticating passwords.

1.4 Extension Testing

We have conducted both manual and automatic testing to verify the functionality of our implementation.

We had a file **runtest.c** which ran automatic tests for some of our utility functions. We also manually ran our program with numerous variations in menu and orders to check that it worked as intended. To speed up testing, we also made text files which would simulate user input and passed it as an argument into our main program.

These methods have proven to be greatly effective, as they have helped us to discover and debug several errors in our implementation. For example, while running the program manually, we discovered that we initially freed up memory too early after editing an unpaid order and this resulted in a segmentation fault when the merchant tries to close the transaction. Additionally, our implementation previously had difficulties pinging the email server while operating on Mac OS due to permission issues with Python.

One error that our test suite revealed was that our pepper function initially changed the last character of the user input password instead of appending a random character to the end of it. It was a lot more time-efficient than manual testing, especially when we had to test our code repeatedly while editing. However, we were unable to test registering using a text file as it involved prompting the user to move a menu.txt file.

References

[1] SAP BrandVoice: The Business Case For Eliminating Paper Receipts
`https://www.forbes.com/sites/sap/2019/12/16/eliminating-paper-receipts/#3ee4431e5b97`

[2] NordPass: Password Peppering
`https://nordpass.com/blog/pepper-password/`