CPT109 assessment2 report using SDP

Problem statement/Specification

1. Control invalid input
2. Register, login, and logout has several exceptions should be handled
3. Remove specific data form a file
4. Game
5. Timekeeping
6. Print neat game history
7. Back to main menu at any stage.
8. UI
9. Beautify the display

I split this system design into nine parts. The Analysis and Design section of this report will strictly follow these nine viewpoints.

Analysis

1. Input should be firstly controlled by its size, then determine if it is the required input for the program needs.
2. In addition to the illegal input mentioned above, it is also necessary to detect the existence of an ID and wrong password.
3. There is no function in C to remove data in the file, so needed to find another way.
4. Using what to represent game's data structure, and how to shuffle it. In addition, exception of repeated selections and end-of-game judgements need to be taken seriously.
5. <time.h> is sufficient for this program.
6. Each column of data must be of equal length, but the width must not be less than the maximum data length.
7. All other stages are easy to implement the return main menu function, but how to separate user-selected game options from exit commands during gameplay needs to be considered.
8. Program should have sufficient hints and implies given to user such that user can have a clear idea what to do next instead of missing in the program.

9. Color scheme, window size, specify window position to display specific content, favorable ASCII art, and animation effects.

Design

1. Because fgets() can specify maximum number of data to read, so using to read input from stdin is safe than other function. I specify the maximum number of data plus two is the size I want fgets to read. fgets() will stop reading in maximum plus one in order to give the input \0 for safety purpose, and another extra room is for me to judge whether the input is exceed the limitation. fgets() also read \n, this character is useless and will cause problem when verify input , so I replace it by \0 at first stage.
2. Just a few ifs to handle the exceptions, no need to take effort to design.
3. Use a temporary file to store needed data temporarily; transfer the data from the temporary file after emptying the original file.
4. Using one (or two, but this will complex the program) dimensional array to represent the game's data structure, and using rand() to shuffle. Use if to determine if the selection is repeated, and nest the loop outside to ask the user to re-enter. Define a value for unsolved, and determine if there is a value in the array representing unsolved at the end of each user selection. If there is none, it means the end of the game.
5. Using struct tm to convert time into standard format.
6. The symbol -* in printf() and sizeof(stuctName.member) is powerful enough for formatted output.
7. Specify a special integer for user to exit in the middle of the game. 0 is not a wise choice because when an assignment fails, there is a high probability that the variable will be 0. For example, if the user only types an Enter and the assignment does not succeed, the variable is defaulted to 0. In addition to specifying special values for the unsolved, it should also detect whether the user wants to exit after each user input.
8. I'm not a good graphic designer, and I know nothing about design principles or theories.
9. Because the program is run on the console of windows OS, so I use DOS command and Win API to control the display. I introduced my latest favorite anime character, **Rina Tennouji**, as the game's main character, using **emoticon** as the game's graphics, in the hope that those who play my game will understand her appeal.

Implementation

Source code file name: **1929836_2**

Testing

Both valid and invalid input has been controlled by two function, inputInt() and inputString, defined in IO.c.

Too many illegal inputs can cause hints to be swiped and even crash the console due to poorly thought-out functions design (inputInt() and inputString).

When registering, the expectation that ID has been registered is controlled.

When login, Account does not exist and password does not match are controlled.

The function destoryAccount() is not good enough. Malicious removal of .accounts or .gamehistories may result in data residuals causing anomalies.

When the game is running, the repeat selection exception is controlled.

The program provides the ability to return to the previous menu at all stages.

Appendix

I attached my reference list in my source code.

A video of one Rina Tennoji' performance I wanted to share: https://www.youtube.com/watch?v=u92YYzZnBYM&ab_channel=HenryL.