Week 3, Lab Work 2: Modularizing Your Projects

Task
Select one software project you have worked on which has at least 5 procedures or subroutines/functions. (You may also find one from any open source platform.)

Selected Software Project
Jog Squad

Guide Question 1
Briefly describe the purpose of the project/program.

Jog Squad is a CLI program written in C that functions as a jogging tracker for squads, hence the name. Users can log in or sign up as members of a jogging squad, view individual and team statistics, and access a hall of achievements showcasing squad members' milestones.

Guide Question 2 & 3
Describe the purpose of each subroutine/function. Describe how each procedure relates to the others, if any.

Jog Squad has several subroutines/functions in its codebase. Below are five functions implemented in the system along with its purpose

1. void log_in():
    - When called, this function displays the following interface:
        - Shows the list of registered jogger usernames to choose from in order to log in and access personal jogger information.
    - Only through logging in would the following be accessible
        - view jogger profile
        - log a jogging entry
        - and update jogger age, height, and weight

2. void disp_joggerUN(const char *csv_file):
    - Used in log_in() and indiv_stats() interfaces
    - When called, reads the specified CSV file and displays the jogger username column in order
    - Returns the size of the jogger username column
        - Returning the list size is important because it serves as an upper bound for certain list options. This ensures that the number entered by users does not exceed the actual number of available options.

3. void csvrow_read_signup(const char *csv_file, struct jogger_info j[MAX_MEMBERS]):
    - Used in jog_squad_members(), log_in(), store_jogentry() and other interfaces
    - When called, reads the specified CSV file until EOF and stores column information per row into the specified *struct jogger_info*

4. void view_stats():
   - When called, this function displays the following interface:
     - An option to 'view individual stats' or 'view team stats'
   - Serves as a gateway to access the interfaces provided by the functions for individual and team stats.

5. void team_stats(const char *csv_signup, const char *csv_jogentry):
   - When called, this function displays the following interface:
     - Reads both the specified CSV files to display team-related information/statistics such as name, total distance, total time, total calories burned

Guide Question 4
Argue or assert on grounds whether these existing subroutines/functions are modularized well by the concept of separation of concerns. Provide at least three(3) assertions and/or suggestions on the matter. (You may suggest improvements on the modularization of your chosen project/program.)

Despite having no prior knowledge of modularization during the development of this C program, I assert that it already demonstrates a **decent** effort toward applying the concept.

<u>Assertions on Modularization (Separation of Concerns)</u>

1. ***Helper functions already demonstrate proper modularization***
   The project applies the concept of separation of concerns by isolating validation, string handling, and UI into small helper functions. For example, display_ascii() is solely responsible for rendering ASCII art, while valid_pass(), valid_M_or_F(), and similar functions handle specific input validation tasks.

2. ***Core functions act primarily as controllers***
   Functions like sign_up() and log_in() delegate tasks such as validation, ASCII rendering, and file I/O to specialized helpers. This is a good sign of modularization, since these functions now serve as controllers that coordinate the process rather than embedding all logic in a single block. This structure aligns with separation of concerns because each responsibility is handled at the correct layer.

3. ***Improvements can still be made by reducing coupling and restructuring files***
   While concerns are well-separated at the function level, the program currently uses global variables (e.g., struct jogger_info jogger[MAX_MEMBERS]) which introduces hidden dependencies. Passing state explicitly would improve modularity. Furthermore, grouping functions into separate source files (e.g., ui.c, validation.c, csv_utils.c) would reinforce separation at the project structure level, making the codebase more maintainable and scalable.