Program Design Project 8

The program `roster.c` maintains a roster for a youth orchestra program. Each student is stored with last name, first name, email, instrument, and group. Complete the program so it uses a dynamically allocated linked list to store the roster and has the following functions:

1. `add`: ask the user to enter a student's last name, first name, email, instrument, and group (in the exact order), then add the student to the **end** of the linked list.
    a. It should check whether the student has already existed by email and last name. If so, the function should print a message and exit.
    b. If the student does not exist, allocate memory for the student, store the data, and add the student to the end of the linked list.
    c. If the list is empty, the function should return the pointer to the student. Otherwise, add the student to the end of the linked list and return the pointer to the linked list.
    d. Assume all inputs are lower case for alphabetical letters.

2. `search`: find a student by last name and first name, print last name, first name, email, instrument, and group of students matching the last name and first name. If no student is found with the last name and first name, print a message.
3. `print_list`: print the last name, first name, email, instrument, and group of all the students.
4. `clear_list`: when the user exists the program, all the memory allocated for the linked list should be deallocated.
5. Use read_line function included in the program for reading in strings.

Testing and submission guidelines:

1. Download the test script from Canvas and upload them to the **student cluster (sc.rc.usf.edu)**. Change the file permissions of the test script with the following command:

    chmod +x try _project8_roster

2. Compile and test your solution for Project 8:

    gcc -Wall project8_roster.c

    ./try_project8_roster

3. Download the program from student cluster and submit it on Canvas>Assignments->Project 8.

Grading

Total points: 100

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%
    a. Function implementation meets the requirements
    b. Function processes the linked list using **`malloc`** and **`free`** functions properly

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
4. Use consistent indentation to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **#define PI 3.141592**
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use underscores to make compound names easier to read: **tot_vol** or **total_volumn** is clearer than totalvolumn.