Project 9, Program Design

1. (50 points) Add a function in `project8_roster.c` that removes a student from the list. The function has the following prototype:

   ```
   struct student* remove_from_list(struct student *list);
   ```

   You will also need to add the function prototype to the program; modify the main function to add `'r'` for delete option in the menu and it calls the delete function when the `'r'` option is selected.

   a. The program asks the user to enter last name, first name, and email of the student to be removed from the list.
   b. Removing a node from a linked list involves three steps: Locate the node to be removed; alter the previous node so that it "bypasses" the removed node; call free to reclaim the space occupied by the removed node.
   c. If the student entered does not exist, print a message and return the list.
   d. Name your program project9_roster.c.

2. (50 points) Modify project 7 so that it uses quick sort to sort the states. Instead of using the sorting function you wrote for project 7, use the quick sort library function and implement the comparison function. Name your file *project9_crimes.c.*

Testing and submission:

1. Download the test scripts 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_project9_roster

   chmod +x try_project9_crimes

2. Compile and test your solution:

   gcc -Wall project9_roster.c

   ./try_project9_roster

gcc –Wall project9_crimes.c

./try_project9_crimes

Submission instructions:

Download the source files (.c) from student cluster and submit it on Canvas>Assignments->Project 9.

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.