Project 12, Program Design

1. (70 points) Modify project 11 by adding and modifying the following functions:

    1) Add a delete function in `player.c` that delete a player from the list. The function should delete a player by the player number. The player number will be entered by the user. The function should have the following prototype:

    ```
    struct player* delete_from_list(struct player
    *roster);
    ```

    You will also need to add the function prototype to the header file; modify the main function in `roster.c` to add `'d'` for delete option in the menu and it calls the delete function when the `'d'` option is selected.

    2) Modify the append_to_list function so the player is inserted into an ordered list (by last name and first name) and the list remains ordered after the insertion. For example, `Joe Martin` should be after `Doug Martin` but before `Jameis Winston` in the list.

2. (30 points) Write a program `sort_commands.c` that sorts a series of words as command-line arguments. For example, running the program by typing

```
./inOrder hello darkness my old friend
```

should produce the following output:

```
darkness friend hello my old
```

Sort the array of strings from the command line using `qsort` and then print the words in a sorted order. Note: the array of string to be sorted should not include the program name such as ./a.out or ./inOrder.

Total points: 100 (60 points for part 1 and 40 points for part 2)

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. **Implementation meets the requirement.**
    b. **Using the malloc and free function properly.**

Before you submit

1. (part 1) Compile with makefile. Be sure it compiles on ***circe*** with no errors and no warnings.

2. (part 1) Test your program with script *try_roster* (It's updated for project 12)

   chmod +x try_roster
   ./try_roster

3. (part 2) Compile your program with the following command:

   gcc –Wall -o inOrder sort_commands.c

4. (part 2) Test your program with *try_words* program to test part 2.

   chmod +x try_words
   ./try_words

5. Your source files should be read & write protected. Change file permission on Unix using chmod 600.

6. Submit all the source files, header files, and makefile for part 1 and *sort_commands.c* for part 2 on Canvas.

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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.