Project 10, Program Design

1. (60 points) A daycare center would like to maintain a list of book requests from the teachers for their classroom libraries. Each book was stored with the title, author's first name, last name, price, and classroom. Modify `add_to_end` function in `book.c` in Project 9:

    a. Instead of adding to the end of the linked list, a book is added to an ordered linked list. A book is added to an **ordered linked list** by author's last name and then by author's first name. The list remains ordered after the new request is added. If two books have the same author, the order doesn't matter. For example, a book by `Ashley Spires` should be before a book by `Whee Winn` in the list; a book by `Elizabeth Spires` should be after a book by `Ashely Spires` in the list.

2. (40 points) Modify project 7 so that it uses quick sort to sorts the cars by city mpg. Name your program `car2.c`. Instead of using the sorting function you wrote for project 7, use the quick sort library function and implement the comparison function.

Before you submit

1. (part 1) Test your program with testing script. This script assumes the executable of your program is named `book_requests`.

    chmod +x try_requests2

    ./try_requests2

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

    gcc –Wall cars2.c

3. (part 2) Test your program.

    chmod +x try_cars

    ./try_cars

4. Submit `book.c, book.h, readline.c, readline.h, book_requests.c, try_requests2,` and `makefile` in a zipped folder.

5. For part 2, submit *cars2.c* and *cars.csv* (for grading purpose).

Grading

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%

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 totalvolu