Project 10, Program Design

1. (50 points) Implement the following function in the `rec_center2.c` program provided:

   `add` function:
   a. Ask the user to enter the student email address, class, first name, and last name **(In the exact order).**
   b. Allocate memory for the structure, store the data, and **add it to the ordered linked list.**
   c. If the list is empty, the function should return the pointer to the newly created linked list.
   d. If the list is not empty, add the request to the ordered linked list and return the pointer to the linked list. A request is added to an **ordered linked list** by class and then by last name. The list remains ordered after the new request is added. You can assume that no two requests have the same class and student name.

      For example, a request for dance class by `Ashley Meyers` should be added before a request for tennis class by `Justin Lewis` in the list; a request for tennis class by `Ashley Meyers` should be added after a request for tennis class by `Justin Lewis` in the list.

2. (50 points) Modify project 7 so that it uses quick sort to sorts the animals in age. Name your program animal2.c. Instead of using the sorting function you wrote for project 7, use the quick sort library function and implement the comparison function.
   Note: For animals with the same age, the order of the animals does not matter.

Total points: 100 (50 points for part 1 and 50 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 properly.**

Before you submit

1. (part 1) Test your program with script *try_rec2.*

   chmod +x try_rec2
   ./try_rec2

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

   ```
   gcc –Wall animals2.c
   ```

3. (part 2) Test your program.

   ```
   chmod +x try_animals
   ./try_animals
   ```

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

5. Submit *rec_center2.c* for part 1 and *animals2.c* and *animals.txt* (for grading purpose) 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 totalvolu