Project 8, Program Design

A recreation center offers dance, art, and other classes to seniors. Write a program that helps the center to manage their waiting lists for these classes. This program loads the current waiting list from `waiting_list.txt`, stores new requests using a linked list, and saves the updated list in the same file.

Each request was represented by the student contact email (a string), class name (a string), student's last name (a string), and student's first name (a string).

The provided program *`rec_classes.c`* contains the *`struct request`* declaration, function prototypes, the main function, and the load and save functions. Do not modify these functions.

Complete the following function definitions so it uses a **dynamically allocated linked list** to store the waiting list requests.

1. `delete`: In this function, ask the user to enter the student first name and last name, email address, and the class name, the function finds the matching request and remove it from the linked list. If it is not found, print a message.
2. `append`:
    a. Ask the user to enter the student email address, class name, first name, and last name. Check whether a request has already existed by all of the information. If a request has the same student first name, last name, email address, and class name as an existing request in the list, the function should print a message about request already existed in the waiting list and exit.
    b. If the request does not exist, allocate memory for the structure, store the data, and **append it to (add to the end of) the linked list.**
    c. If the list is empty, the function should return the pointer to the newly created linked list. Otherwise, add the request to the end of the linked list and return the pointer to the linked list.

3. `clearList`: when the user exists the program, all the memory allocated for the linked list should be deallocated.

Note: use read_line function included in the program for reading a line of text.

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 requirement.**
    b. **Function process the linked list by using the malloc and free functions properly.**

Before you submit

1. Compile with –Wall. Be sure it compiles on *the student cluster* with no errors and no warnings.

gcc –Wall rec_classes.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

chmod 600 rec_classes.c

3. **Test your program manually before using the Unix Shell script. `waiting_list.txt` *should be in the same folder as try_rec when you test your program.***

chmod +x try_rec

./try_rec

Note: Your program will modify waiting_list.txt when it runs, so you will need to reload the original waiting_list.txt to your directory next time you test your program.

4. Submit *rec_classes.c* and waiting_list.txt 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.