Project 8, Program Design

Write a program that helps a daycare center manage their waiting list. The program stores requests for being added to the waiting list. Each request was stored with the classroom, the child's first name, last name, and contact phone number. The provided program *waiting_list.c* contains the *struct request* declaration, function prototypes, and the main function. Complete the function definitions so it uses a **dynamically allocated linked list** to store the waiting list requests. Complete the following functions:

1. **append_to_list:**
    a. Ask the user to enter the classroom, child's first name, and last name.
    b. Check whether a request has already existed by classroom and the child's first name and last name. If a request has the same classroom, first name and last name as an existing request in the list, the function should print a message about using the update function to update classroom and exit.
    c. If the request does not exist, ask the user to enter the contact phone number.
    d. Allocate memory for the structure, store the data, and **append it to (add to the end of) the linked list.**
    e. 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.

2. **update:** update a request's classroom. When a child is on the waiting list for a long period of time, he/she might need to be moved to the waiting list for another classroom. In this function, ask the user to enter the classroom, child's first name, and last name. Find the matching child, ask the user to enter the new classroom and update the classroom. If the child is not found, print a message.
3. `printList`: print the classroom, child's first and last name, and contact phone number of all requests in the list.
4. `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 first name, last name, classroom, and phone number.

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 waiting_list.c

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

chmod 600 waiting_list.c

3. Test your program with Unix Shell script

chmod +x try_list

./try_list

4. Submit *waiting_list.c* 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.