Project 10

The program `dogs.c` maintains records for canine patients at an animal hospital. Each dog's record has a name, a breed, a patient number, and owner's last name. Complete the program so it uses a **dynamically allocated linked list** to store the records and contains the following functions:

1. `append`: ask the user to enter patient number, dog's name, dog's breed, and owner's last name, then add the player to the **end** of the linked list.
    a. It should check whether the dog has already existed by patient number. If so, the function should print a message and exit.
    b. If the dog does not exist, allocate memory for the dog, store the data, and append the dog to the end of the linked list.
    c. If the list is empty, the function should return the pointer to the newly created dog.
    d. Otherwise, add the dog to the end of the linked list and return the pointer to the linked list.

2. `search`: find the dog by name, print all the dog's information that matches the name. If the dog is not found, print a message.
3. `print`: print the name and number of all the dogs.
4. `clear`: 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 in dog names, breeds, and owner last names.

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 function properly.**

Before you submit

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

gcc –Wall dogs.c

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

chmod 600 dogs.c

3. Test your program with Unix Shell script try_roster

chmod +x try_dogs

./try_dogs

4. Submit dogs.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.