

Project 8, Program Design

A small car dealer would like to maintain information for the cars in stock. Each car was stored with the make and model, color, manufacture year, city mpg, highway mpg, and quantity. The program *dealer.c* contains the *car struct* declaration, function prototypes, and the main function. Complete the function definitions so it uses a dynamically allocated linked list to store the cars. The following functions should be completed:

1. `append_to_list`: ask the user to enter a car's make, model, color, manufacture year, city mpg, highway mpg, and quantity (in the exact order), then add the car to the **end** of the linked list.
 - a. It should check whether the car has already existed by make, model, color, and manufacture year. If so, the function should print a message and exit.
 - b. If the car does not exist, allocate memory for the player, store the data, and append the car to the end of the linked list.
 - c. If the list is empty, the function should return the pointer to the newly created car.
 - d. Otherwise, add the car to the end of the linked list and return the pointer to the linked list.
2. `find_car`: search by make and model, print the car's color, manufacture year, city mpg, highway mpg, and quantity. Print all cars that match the make and model. If the make and model is not found, print a message.
3. `printList`: print the make, model, and manufacture year, color, city mpg, highway mpg, and quantity of all the cars.
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 in make, model and color. **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 dealer.c`

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

`chmod 600 dealer.c`

3. Test your program with Unix Shell script `try_cars`

`chmod +x try_cars`

`./try_cars`

4. Submit *dealer.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`.