Mercedes Car Exhibition

Assignment learning objectives: You are supposed to correctly define and write neat code that uses queues.

The assignment problem:

For the last assignment of the semester, you need to develop a simple cars' management system for the Mercedes exhibition. The exhibition has 3 types: fast cars, old cars, and luxury cars. The first screen of your application will ask the user to select one of the car types (see figure 1). You need to create a queue for each type that includes cars of the current type only. Users of your software can do the following functions within each cars type (see figure 2):

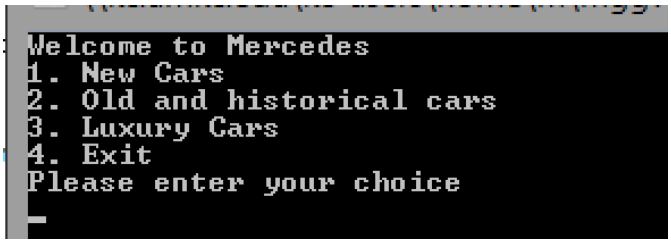

Figure 1. The application's main screen

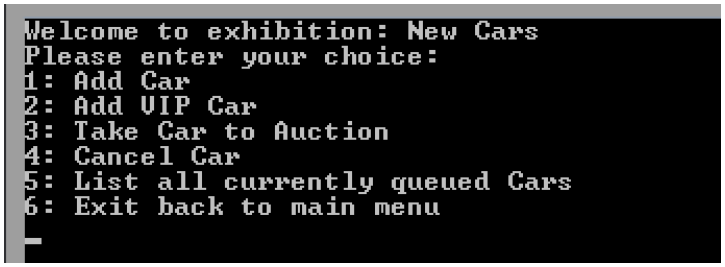

Figure 2. The screen of each type

- **Add new car**

The "*Add car*" is responsible to add a new car entry into the queue. Each car must have a name, model, and a plate number (cannot be empty, but can be of any length for simplicity, e.g. 123 is accepted). If the plate number is empty (return character), the operation should be canceled with cancellation message. The maximum number of cars for any type is 100. Note that after adding a new car, your application should confirm the entered data of the car on a *new screen*. After confirming the entered data, the user can press any key to return to the cars' types menu. You do not need to ask the user if the entered data is correct or not. Just confirm it by re-printing it.

- **Add VIP cars (very expensive cars)**

The "*Add VIP cars*" will add a new car (name, model, plate number) at the beginning of the current queue – passing all current cars in the queue. This means the car which was entered the queue as VIP car will have a higher priority than other cars, even if they were entered the queue after the regular cars. Your application can add any number of cars as VIP cars, but not more than 100 cars in total within each type for both regular and VIP cars. Note that, if you have several VIP cars then the priority is the same as they entered the queue (First in, first

out). Once all VIP cars have been taken out to the auction (removed from the queue), the application continues with the regular cars –also in a first in, first out manner. Note that after adding a new car, your application should confirm the entered data of the car on a *new screen*. After confirming the entered data, the user can press any key to return to the types' menu. You do not need to ask the user if the entered data is correct or not. Just confirm it by re-printing it.

- **Take out a car for auction**

This feature is responsible to move cars to the auction place (remove them from the queue). Of course, this should be a first-come first-served operation. Note that having VIP cars will give them higher priority to enter the auction first (to be removed from the queue first, even if they have been added after the regular cars). After moving a car to the auction, your application should print the data of the moved car (name, model, and plate number) in a new screen. If no cars are left, your application should print a notification message to the screen "No more cars."

- **Cancel car**

This feature will ask the user to input the data of a car (name, model, and plate number) and then remove that car from the queue. If such a car does not exist, a notification message is printed to the screen "Car Does Not Exist."

- **List all current cars**

This feature lists all cars currently in the queue (name, model, plate number) including regular and VIP cars.

- **Exit the current cars' type**

This function exists the current type, clear the screen, and print the main application menu (Figure 1).

You need to define a struct called *Car* to define cars using first name, model, and plate number. In addition, your application must have a class called *queue* that includes the aforementioned functions (methods) as member functions of the class such as, AddCar(); AddCarEnd(); AddCarBeginning(); CancelCar(); etc. Feel free to create your own methods –these are just examples.

Submission:

In order to receive full credit, your program must be running correctly, implements all required functions/data and organized.