Auto Inventory System – Final Project

**Formalized Proposal**

Many small auto dealerships still manually keep track of their inventory.  Sales Consultants may have a paper list of inventory that is outdated. It may not reflect new inventory, sold inventory, or price changes. A digital inventory system would allow these small dealerships to keep track of current inventory and prices.

Auto Inventory System will use a Linked List of cars, indicating the stock number, year, make, model, color, mileage, and price. Managers and Sales will be able to add cars, update cars, delete from inventory, search for a car, and show a list of all inventory.

For the search function, the user will be able to choose how they want it sorted (by stock number, year, make, model, or color).

**Time/Change Logs**

My first week, I sketched out pseudocode to see what all I wanted to accomplish with the program. I completed all the initial coding for the main driver and the methods to enter, update, search, view, and delete information. When I first started writing my code for the application, I realized I was going to reuse a lot of code as I created my choice pathways. As a result, I wrote methods for most of the code I knew I would be using regularly. I was then able to just call the method in my main program every time I needed to use the code.  I felt that would be the cleanest way to code my program but needed a refresher from my C# class assignments. I also googled library methods I could use to accomplish what I wanted. I had to do research on Column formatting and switch cases. I had initially been using if/else code for my choices but found I was having a terrible time keeping track of my pathways as I coded them. I had remembered briefly touching on cases in my C# class and spent time researching how to write them. I updated all my choices to cases which tidied of a lot of my code.

My second week, I expanded on the coding for the search option. I initially had started with just the stock number to search, but I expanded it to include searching by year, make, model, and color as well. I also updated some of the visuals on the program and included more formatting. I made changes to my coding to include nested cases and updated my search case coding.

My third week, I implemented a foreach loop to write out multiple results from my list for my search option. I removed the return car coding I had initially used and converted my search methods to void.

My fourth week, I wrote the unit tests and ran the program multiple times looking for any additional program errors. I found a glitch that my search function would return an empty list if there were no results.

My fifth week, I figured out how to have the program return “No matching inventory found.” if my search came back empty.

**Lessons Learned**

My project began as a barebones inventory list and expanded to include a more user-friendly experience. There were initially limited options on my search function, and I struggled to set up the code to allow for mistakes in user input. I was able to expand my search function to include different ways to search for inventory and make the program easier to navigate back and forth.

My first week, I had issues with research in general with C#. Whenever I would use google for research, I would easily find examples for C, C++, and Java code. I had to spend much more time than I liked researching for this project.

My second week, I struggled to set up my code in a way that allowed the search function to return more than the first results it found. Also, I had set it up to search by stock number because no two cars should have the same stock number. But what if you don't know what car you're looking for yet? You wouldn't have the stock number, so I had to set it up to search by year, make, model, and color as well. However, my Find() method only returned the first result found, not all results, and the FindAll() method wouldn't work. No matter how I coded it, it was an error, so I had to do a lot of searching to find alternative methods. I then researched how to print them out and how to format columns, so the results were in a neat, readable format. I was eventually able to find some examples that allowed me to write my own foreach loop.

My third week, I was sick most of the week. I was having problems writing my Unit Testing and wasn’t finding much on Google to help me with it. Everything I did find indicated void methods with no arguments couldn’t be tested. I created the tests for the methods I did have arguments for.

My fourth week, I found a few additional glitches in my coding. I had to set up my delete option to show the details of the car they had selected and so the user could be sure it was the correct car before allowing the user to delete the car. This prevented any issues with accidentally typing in a wrong stock number and having it automatically deleted. I added the same code to my update function, so they’d be sure it was the correct car they were updating.

**CODE including comments**

You can find my final project at <https://github.com/buttery85/Data-Structures-Buttery>.

**User's Manual**

Select Your Choice:

* + 1. Add a Car
       - Enter Stock Number
       - Enter Year
       - Enter Make
       - Enter Model
       - Enter Color
       - Enter Mileage
       - Enter Price
       - Do you want to continue with the application?
    2. Update a Car
       - Enter Stock Number
       - Confirm if it’s the car you want to update.
       - Select Your Choice

1. Update Stock Number

2. Update Year

3. Update Make

4. Update Model

5. Update Color

6. Update Mileage

7. Update Price

* + - * Enter Change
      * Anything else you want to update on this car?
      * If no, do you want to continue with the application?
    1. Delete a Car
       - Enter Stock Number
       - Confirm it’s the car you want to delete.
       - If yes, the car is deleted.
       - Do you want to continue with the application?
    2. Search for a Car
       - Select Your Choice
         1. By Stock Number

Enter stock Number.

* + - * 1. By Year

Enter Year

* + - * 1. By Make

Enter Make

* + - * 1. By Model

Enter Model

* + - * 1. By Color

Enter Color

* + - * Do you want to continue with the application?
    1. View a Full List of Cars
    2. Exit

**Conclusion/Summary**

I utilized the MERUSE principles of good programming for my code.

**Modularity** – A CarList class was added to encapsulate the data.

**Efficiency** – I made the search and update features easy to navigate. As the lists are generally smaller with small dealerships, I used a straightforward foreach loop to write the list out.

**Robustness** – The program was run many times to catch any errors in code. Each error was corrected, and the program should run now without any errors.

**Usability** – The program is user friendly and easy to understand. It requires minimal instruction.

**Should be** **Readable** – All code was set up with easy-to-understand methods and variables. Example: ViewAllCArs() method is the code written to view a full list of the cars.

**Elegant** – My code started out with a lot of if/else code and evolved to case code to be more efficient and easier to read. Methods for each task were written to segment the code into easy to find and understand segments.

In summary, the Auto Inventory System is an intuitive, interactive program. It allows users to add, update, delete, search, and view a complete list of cars. It is limited to run time at the moment. Future versions would implement an SQL database to keep a permanent record of the list upon opening and closing of the application. The GUI would also be updated to run the program via an internet website using the web application feature of Visual Studio. Pictures of the inventory would be included, and links would be added to access them.