Project 2 Documentation

Purpose of the Program

==================

This is a program that is meant to manipulate data for all cars from an input file. After getting the name of the file, the program will display an interactive menu that displays various options for the user to select that lead to different functionalities being executed on the input file data. Specifically, it reads in all the data of each car in the input file using the input file stream as structs of data for each car. Each of these cars, which each represent one struct, is then stored into an array of structs, after which the program will manipulate the data in various ways such as printing data, searching for certain cars, and calculating cost for rentals using an interactive menu that relies on user input. Essentially, the program acts as a menu program that gives the user various choices at manipulating a database of cars based on their traits.

Program Design

=============

The program initializes a new struct data type RentalCar to store the data for each car being read in, with each struct capable of storing various bits of information about a car from its make, year, and model to its rental price per day and availability. The program also initializes an array of structs to read in the data of multiple cars from the input file, storing them together. First, the program asks for the user to enter the name of their desired input file name through prompts. The program then inputs the name entered into a character array representing the file name and opens the input file.

The program will then display an interactive menu with options that allow for different functionalities to be executed on the car data based on user inputs. These functionalities will include reading all the car data from the file, printing out all the data to the terminal and a separate output file, estimating car rental cost given that the user provides as input a car number and the number of days the car will be rented out, finding the most expensive car and printing all its data, printing out data for only all the available cars to both the terminal and a separate output file, and exiting the program once the user finishes manipulating the car data. If a user selects to read in all data, the program reads in all the data for all cars listed in the input file using the input file stream by placing the data for each car as a struct of type RentalCar into the array of structs storing all car data. If a user selects to print all data, the program will print out the data for each car that is stored in the RentalCar array both to the terminal and a separate output file that is titled “AllCars.txt” in the same format as the input file. If a user selects to estimate the total rental cost of a car, the program will prompt the user for a specific car number and number of days of renting the car. After receiving these inputs, the program then calculates the total cost for renting the specific car in dollars. If a user selects to find the most expensive car, the program will search for the most expensive car based on its rental price per day and print out all its data to the terminal. If a user selects to print out available cars, the program will print out data for all available cars both to the terminal and a separate output file that is titled “AvailableCars.txt.” Finally, if a user selects to end the program, the entire program will terminate. After each menu option selection, if the user has not selected to exit the program, the program will always repeat the interactive menu options prompts to the terminal, requesting the user for further inputs.

Changes to Program

================

I would not have made any changes to the program and its functionalities since I believe that I made the program robust enough to read in all sorts of input files, regardless of how much data the program must read in for however many number of cars represented in the input file. The separation of menu options into their own functions is also a good feature that I would leave untouched because it performs to the best of the program’s capabilities. If there was anything I would have changed, it most likely would have been to make the callMenu function more streamlined and robust at taking in fewer parameters in order to function in a more efficient manner without having to rely on passing too many parameters either by value or by reference methods to function properly and affect the functions downstream.