Project 4 Documentation

Purpose of the Program

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

This is a program that is meant to manipulate data for all cars from one rental agency listed in an input file. After getting the file name, 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 and their agencies in the input file using the input file stream as class objects for the rental agency, each rental car, and each car sensor. Each of these car and sensor objects, which each represent one class, is then stored into arrays of classes, after which the program will then manipulate the data in various ways such as printing all 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.

Program Design

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

The program initializes new Class data types CarSensor, RentalCar, and RentalAgency to store the data for the agency, each car, and each car sensor being read in, with each class capable of storing various bits of information about a car from its make, year, and model to its base and final rental price per day, availability, number of sensors, and what types of sensors. The CarSensor class includes functions that manipulate the data for each sensor, including default, parameterized, and copy constructors along with get and set methods for each member variable, including the four static variable counters for each type of car sensor available, and get and reset functions for each of the static variables that reset them to zero. The RentalCar class also includes functions that can manipulate the data for each car, which are able to print the car’s data, estimate its rental cost given the user provides input on number of days the car is to be rented out, get and set methods for all the car’s data members, update the final rental price based on the sensors the car has, and both a default constructor (which sets initial values for the RentalCar’s data members) and a parameterized constructor (which creates a new object based on the values for the car’s data members passed into it). The RentalCar class also has two overloaded operators that add on a new owner or lessee if the car is rented or a new sensor is added. The RentalAgency class is initialized to store all the data for the rental car agency being read in, capable of storing bits of information about the agency’s name, zip code, and all data for its inventory of cars. The program initializes an array of their 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.

Within the CarSensor class, the GetExtraCost(), GetType(), GetGps(), GetRad(), GetLid(), and GetCam() functions are designed to retrieve the values for their respective data members (i.e. GetType() retrieves value of m\_type) for a specific car object without having to access the data members directly. These functions are necessary for retrieval of values due to the fact that all the variable data members of the CarSensor Class are private, meaning that they cannot be called on directly through variable name in the code. Similarly, the SetExtraCost(), SetType(), SetGps(), SetRad(), SetLid(), and SetCam() functions also affect their respective data members by setting their values to new values that are passed into the functions when the functions are called. Finally, besides the constructors that were described before, the RentalCar class contains the GetGpsReset(), GetRadReset(), GetLidReset(), and GetCamReset() functions. These functions collectively reset their respective static member variables to zero, the initial blank value for these counters without increments. All functions in CarSensor are public. Just outside the function there is also an operator== overloaded to compare two CarSensor objects.

In the RentalCar class, the getYear(), getMake(), getModel(),getTank(), getBasePrice(), getFinalPrice(), getOwner(), getSnscnt(), and getAvailable() functions are designed to retrieve the values for their respective data members (i.e. getYear() retrieves value of m\_year) for a specific car object without having to access the data members directly. These functions are necessary for retrieval of values due to the fact that all the variable data members of the RentalCar Class are private, meaning that they cannot be called on directly through variable name in the code. Similarly, all of the SetYear(), SetMake(), SetModel(), SetOwner(), SetBasePrice(),SetSnsCnt(), SetTank(), UpdatePrice(), and SetAvailable() functions also affect their respective data members by setting their values to new values that are passed into the functions when the functions are called. Finally, besides the constructors that were described before, the RentalCar class contains the PrintCar() and EstimateCost() functions. The PrintCar() function, given the output stream variable and the pointer to a car’s data, prints out all the data for that specified car object. The EstimateCost() function, given the pointer to a car’s data and the number of days to rent out that car as requested by user input, will return the value of the total rental cost for that specified car. There are also two in-class operator+ functions overloaded to add a lessee to a car or a new sensor to a rental car. All functions in RentalCar are public.

The RentalAgency class has similar structure to the RentalAgency struct, with arrays for the zipcode, name of the agency, and inventory of all cars at the agency. Besides the agency constructor, the class also contains functions that manipulate the data, with get and set methods for all member variables except for the inventory, which has its own GetInventoryItem operator[] for reading and writing to the agency’s car inventory similar to the RentalAgency struct in the last project.

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 data for the agencies and their cars from the file, printing out all the agency and car data to the terminal and a separate output file, estimating car rental cost given that the user provides as input the name of the rental agency, a car number, and the number of days the car will be rented out, finding the most expensive car from all three agencies and printing all its data, printing out data for only all the available cars from all the agencies to both the terminal and a separate output file, and exiting the program once the user finishes manipulating the agency and car data. If a user selects to read in all data, the program reads in all the data for all agencies and their cars listed in the input file using the input file stream by placing the data for each agency as a struct of type RentalAgency into the array of structs storing all agency data through the use of a pointer for the agencies struct array. An int pointer will manipulate data for the read in of the zip code for each agency, helping to store each of the zip code’s digits into an element of an int array that is associated with its agency. For each struct of each agency, an array of classes of type RentalCar will store all the car data and the information for the class’s functions, including prototypes and parameters. For each car, the program will read in the values for the RentalCar class object’s individual data members into temporary variables, which will be passed into their respective set methods through manipulation of a pointer that is directed to the array of RentalCar class objects within each RentalAgency struct. After each iteration of their arrays, the pointers will increment and move to the next element of their array to read it in. The exception is when the RentalAgency pointer increments, upon which the RentalCar pointer will point to the first element of the new agency’s inventory array of car objects and the int pointer will point to the first element of the int array that represents the new agency’s zip code.

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 in the same format as the input file. The program does this in a similar fashion of pointer manipulation as in the readIn() function. Specifically, it initializes the RentalAgency pointer to the first element of the agencies array, a RentalCar pointer to the first element of the first agency’s inventory array of cars, and an int pointer to the first element of the first agency’s zip code int array. For each agency, the program will then output the name of the agency followed by its zip code, and all the data for all the car objects inside its own inventory. After each iteration of their respective arrays, the pointers will increment to move on to the next element and repeat the process, with each increment of the RentalAgency pointer leading to the process of reinitializing both the RentalCar and int pointers to their arrays in the struct of the new agency.

If a user selects to estimate the total rental cost of a car, a RentalAgency and RentalCar pointer are both initialized to the first elements of their respective arrays like in other functionalities. A counter for the car number in an agency inventory is also initialized. The program will then prompt the user for the name of the rental agency from which he wants to rent the car while also listing the available agencies to choose from. The user is then supposed to select from these available agencies and put in the choice as input. The program will then prompt the user for a specific car number and the number of days of renting the car. After receiving these inputs, the program uses the strcmp() function to compare the user’s input for the agency name with each of the available agencies’ names. If for a certain agency name it does not find a match, the program increments the RentalAgency pointer and then moves on to the next agency name. Once it finds the correct agency name, the program sets the RentalCar pointer to the first element of the car inventory for the matched agency. At each element of the car inventory array, the value for the car number from user input is compared to the current car number counter for equality. If they are not equal, the RentalCar pointer will increment to the next element in the inventory while the counter also increases. Once the correct car is found, the EstimateCost function takes the RentalCar pointer to have access to the car’s data and number of days for renting the car from user input to return the total rental cost.

If a user selects to find the most expensive available car, the program will search for the most expensive car based on its rental price per day from the agency and print out all its data to the terminal. It does this by first initializing two RentalCar pointers and one RentalAgency pointer. One RentalCar pointer points to the most expensive car and the other points to the current car being read in an agency inventory; both are set to the first element of the first agency’s car inventory at first. The program then iterates through every one of the agencies and each car in their inventories. If the price of the current car exceeds that of the car that is referenced by the pointer for the most expensive car at that moment, the pointer for the most expensive car points to the new car object. A new iteration of a car inventory causes the pointer for the current car to increment, while a new iteration of the agency array causes the RentalAgency pointer to increment while the RentalCar pointer for the current car is set to the first element of the new agency’s inventory. 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. 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 to the program. Also, the function asks the user if he wants to rent the car, and it will update the car’s availability and current owner if the user says yes.

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 the types of input files that it was designed for. The separation of menu options into their own functions is 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. I also would have liked to possibly include an if conditional that would allow the program to respond to improper inputs for a user for the functionality that estimates rental cost with regards to actual name of the rental agency that is requested by printing out an error message and prompting the user to enter a valid agency name as input for menu option 3. I also would