Automobile Dealer Inventory Manager

By Anthony Garrett

Program Design

1. Problem Definition

The local car dealer has been having a hard time keeping up with his inventory. He has decided that he wants to go more modern and keep all his inventory information on his computer instead of in the filing cabinets he is currently using. The owner is looking for a programmer to help by providing a program that will allow him to add cars to his inventory, search the inventory, and display the inventory on the screen. Write a program to help the owner out with his problem.

1. Program Analysis

2.1 User Requirements

The owner of the shop provided information that he required for every car he has in his inventory. He needs the following information for every car that the program stores: Make-and-Model, Year, Color, Class(sedan/truck/van/SUV/minivan), Fuel-Type (diesel, gasoline, electric, hybrid), and Price. When the cars are displayed to the screen he also needs this information to appear on the screen. Therefore, the layout for the output will look as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Make/Model | Year | Color | Class | Fuel Type | Price |
| Dodge Ram | 2005 | Rose Gold | truck | gasoline | $9,456 |
| Ford Taurus | 2018 | Fire Red | sedan | gasoline | $21,354 |
| Jeep Compass | 2016 | Canary Yellow | SUV | diesel | $26,795 |

When the user searches for a specific car from the inventory the information will be displayed in the same formatting with only a single entry in the table for the car that was found in the search.

The initial information that is stored in the inventory array will be data that is read in from a file on the disk that holds the information in such a way that data for each car is stored on one line each. This file will be read at the beginning of the program and will be written to when the program exits.

2.2 Choice of language features, variables, and formulas

I will be using the C++ programming language to complete this program. I have been working on my skills in C++ which will save the time of switching to another language. The C++ language also has a very fast execution time and the owner would be very interested in a speedy inventory program.

I will require the following libraries to complete the program:

Library Usage

iostream processing input and output from the keyboard and console

iomanip Formatting the information as it is displayed to the screen

fstream Reading and writing to files with file stream objects

string Library for functions to manipulate string objects

**Input:** The program will process a single file as input, **carsInStock**. When the program is first executed it will use variable **inputFile** that will be an ifstream variable to read the file contents into the **carInventory** array.

**Output**: The program may process output in two ways. The first method is output that is displayed to the screen for the user. This data will be in tabular form for both the **displayCars** function and the **searchCar** function. The second type of output is a file on disk that will contain all the car inventory. This file will be written to using an ofstream object named **outputFile.** This file will be overwritten every time the program exits and named **carsInStock.**

The following variables will be needed for the main program:

Variable Type Usage

mainInventory AutoDealer instance of class AutoDealer to process the user’s menu choice

user\_input int Number to hold the choice for the user menu

**Formulas**

No specific formulas will be used in the implementation. No calculations will be required as the program will only store car information in a data structure.

2.3 Data Structures

The program will be contained within a class by the name of **AutoDealer**. All the operations that need to be performed on the inventory will take place through methods of the **AutoDealer** class. One important bit of information is how the data is stored in the class. The class will use an array of type **Car**. This will be a struct designed to hold all the information for each individual car in the inventory. Using a class and struct in this manner will allow me to move all the inventory manipulation into the class methods for **AutoDealer**.

This will make the main function of the program much cleaner. Another advantage of using the class to organize this program is that it will provide a custom data type that can be used in the future when developing a similar program for another location that may have slightly different needs. The changes could be made to the class while the main logic stays the same.

The struct **Car** and **AutoDealer** class will organized in the following manner:

1. Design

Algorithm Design

The first thing that needs to happen when the program is executed involves loading up the cars that are already in inventory. This is accomplished by opening the file on disk named **carsInStock** with an ifstream object to read the information from the file into the array of cars in the class. The number of cars in the array will need to be updated as the cars are read into the array. This will be placed in the **carCount** variable. The user may then choose to do one of the following: input new car info, display the cars, search for a car, or exit the program. All the operations will take place in the class methods that will simply need to be called by the instance of the class in the main function. The only logic needed in the main function is the loop containing the user menu and switch statement to process the user’s menu selection.

Function **main**

The following things will occur in the main program:

1. Read all car inventory data from the file on disk named **carsInStock** into the class instance array
2. Allow user input of new car information to store in the array
3. Allow user to search the array for a specific car
4. Display all cars in the array to the screen
5. Repeat 2 – 4 as requested by the user
6. Write the contents of the array to a file on program exit

**Main function pseudocode**

Declare *mainInventory* object

Declare *user\_input* variable

Call *readCarsFile* method of *mainInventory* object

Display user menu until the user enters 4

Get keyboard input for user choice

Switch on user input choice

Choice 1 – Call *inputCarInfo* method of *mainInventory* object

Choice 2 – Call *displayCars* method of *mainInventory* object

Choice 3 – Call *searchCar* method of *mainInventory* object

Choice 4 – Call *writeCars* method of *mainInventory* object

Thank user for using the program

Exit the program

Default – Warn user to pick a valid option

Constructor **AutoDealer**

Default constructor that is called when the class instance is created. The constructor simply sets the **carCount** variable to 0.

**AutoDealer pseudocode**

Set *carCount* to 0

Function **readCarsFile**

This function will create an ifstream object and open the file on disk named **carsInStock** to read in the information that already exists in the file. This information will be read into the array.

**readCarsFile pseudocode**

Declare *inputFile* ifsteam object

Open *carsInStock* file

While not the end of the file

Read car name row into *carInventory[carCount].makeAndModel*

Read car year row into *carInventory[carCount].year*

Read car color row into *carInventory[carCount].color*

Read car class row into *carInventory[carCount].class*

Read car fuelType row into *carInventory[carCount].fuelType*

Read car price row into *carInventory[carCount].price*

Increase carCount by 1

Close the *inputFile*

Function **inputCarInfo**

This function will input the data from the user to populate each piece of information needed to add a new car to the **carInventory** array.

**inputCarInfo pseudocode**

while user doesn’t want to quit

Read car name into *carInventory[carCount].makeAndModel*

Read car year into *carInventory[carCount].year*

Read car color into *carInventory[carCount].color*

Read car class into *carInventory[carCount].class*

Read car fuelType into *carInventory[carCount].fuelType*

Read car price into *carInventory[carCount].price*

Increase *carCount* by 1

Read in user option to add another car

Function **searchCar**

This function will allow the user to find a car that is in the **carInventory** array. If the car is found inside the array the car information will be displayed in a tabular format. If the car isn’t found that will be displayed to the user as well

**searchCar pseudocode**

declare *carMake*, *color*, and *carClass*

Get carMake, color, and class from the keyboard form the user

Declare count variable = 0 and found = 0

While *count* < *carCount*

If *carInventory[count].makeAndModel* is equal to *carMake*

If *carInventory[count].color* is equal to *color*

If *carInventory[count].class* is equal to *carClass*

*Found* = *count*

Break

If found is equal to 0

Display a message that the car was not found

else

Print table headers formatted

Print *carInventory*[found] all information in table columns

Function **displayCars**

This function will read through the entire array of cars that are in **carInventory**. The function will display a table header formatted using the **iomanip** library. The information for all the cars in the array will then be printed under the headers in tabular display format.

**displayCars pseudocode**

print headers with all car attributes

for i at 0 until *carCount* increasing I by 1

print carInventory[i]. of all members in table format

Function **writeCarsFile**

This function will be called when the program is ended by the user. The function will overwrite the **carsInStock** file with the entire contents of the **carInventory** array.

**writeCarsFile pseudocode**

declare *outputFile* ofstream object

open the file *carsInStock*

for i at 0 until I < carCount increasing i by 1

write carInventory[i]. of all members + newline to the file

close outputFile

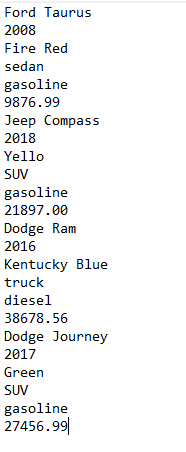
1. Planned Test Data and test environment

4.1 Planned test environment

The program will be developed in the Visual Studio Code text editor on a Linux KDE Neon Distribution machine. The program will be stored on the grace server for BCTC. It will be executed on that server using the GNU g++ compiler. Test files will also be created in the Visual Studio Code editor and saved as the **carsInStock** file that the program is expecting.

4.2 Planned test data

I will be trying a few different files with valid data for the program to read into the array at first execution. One of the files will be like the one below.



After testing the initial file read I will then turn to the main function testing. I will attempt to break the main menu by entering different values as the menu option. I predict that anything other than a number will break the program and cause it to crash or go into an infinite loop. I will also test each of the method calls through the menu options with different input data. One test I will also perform is to display the cars as soon as the program starts, then add a car and try to display the cars again to make sure the new cars are being added to the array. One other thing I will test is to exit the program and make sure the program is writing the new cars to the file properly. Finally, I will then reopen the program to read the file I just wrote on exit and make sure everything works as it should.