CS 474: Object Oriented Programming Languages and Environments
Spring 2017

First C++ project

Due time: **9:00 pm on Sunday 4/9/2017**

For this project you will design and implement a test version of a *Vehicle Inventory System (VCS)* on behalf of *Mr. Tucker Piston* who owns a car dealership business. Tucker has a number of cars (possibly from different years and manufacturers) in his parking lot. He is interested in a system that will allow him to keep track of basic information about each car (e.g., make, model, year) as well accounting information (e.g., date when car was acquired, price paid, and asking price for the car). Thus, each car instance will contain the following fields:

1. *Id*—A unique integer identifier.
2. *Make*—This is the car's manufacturer (e.g., Ford, Honda, etc.)
3. *Model*—This is the car's model (e.g., Explorer, Camry, etc.)
4. *Year*—This is the model year of the car (e.g., 2011).
5. *Date*—This is the day, month and year when the car was acquired.
6. *Cost*—This is the price that Tucker paid for the car.
7. *Picture*—A reference to a file containing a picture of the car in an appropriate format (e.g., jpeg, png or similar format).

For the time being, you do not have to worry about creating, storing or displaying the pictures; you may assume that this functionality will be implemented by someone else. However, you must make provisions for the fact that images will be relatively large, making each car instance expensive to store in memory. For this reason, you decided that you will keep your car instances in files, rather than memory, most of the time. You have also decided to implement a smart pointer class called *CarPtr* that dynamically loads instances of class *Car* when they are needed and stores them back on file when done with a car. Each car instance is stored in a different file; there is a 1-to-1 correspondence between instances of *Car* and *CarPtr*.

You are to code a test version of the program that contains just 5 car instances denoted by unique IDs 1 through 5. The five *CarPtr* instances will be held in memory at all times; however only two of the *Car* instances will be held in memory; the remaining instances will exist only in the files associated with your project.

When the program is started, the program creates a linked list of 5 *CarPtr* instances; however, none of the *Car* instances are loaded in memory. Next, car instances are loaded from memory and stored back to files (possibly after being modified) as needed while executing the commands described next.

Your project should support the command line interface below. Your command line interface will prompt the user for a command, and then execute the command. Here is a list of commands. Make sure not to cause any memory leaks or dangling pointers in the implementation of these commands.

1. `c`—Create a car. This command creates a new car instance and the corresponding *CarPtr* instance. This command is valid only if the user has not entered 5 car instances yet. The user is prompted for all fields contained in the new car instance using interactive prompts. As soon as information about the car is complete, the car instance is saved in a file and deleted from memory. If this command is called when the VCS already holds 5 cars, the command will have no effect.

2. l—List all cars. Each car is loaded from disk and its information (ID, make, etc.) is displayed in the user's screen (except, of course, for the picture). Car instances are stored back on disk as needed to keep a maximum of two total cars in memory.

3. 1 . . . 5—Edit a car. This command sets the numbered car to be the current car. If the car is not already in memory, it is loaded from disk and stored in the corresponding *CarPtr* instance. In this case, a car instance currently in memory may have to be stored back into its corresponding file and deleted from memory to avoid violating the limit on the number of car instances that can held in memory.

4. p—This command changes the asking price of the current car. The user is prompted for a new integer value, which becomes the new asking price. The current car instance is not saved to file as part of this command; however, the new content of the current car is displayed in the user's screen.

5. s—Save all cars. This command save the car instances currently in memory to the corresponding files. Each file is first deleted (e.g., using the *remove()* C++ function). Next, an *ofstream* instance is opened on the named file, and the car's information is saved to the file. Finally, the stream is closed.

6. q—Quits the VCS.

Implementation notes. Your VCS must conform to the following specs:

1. Client code using the VCS will not be allowed to access a car instance directly; these functions are handled by the *CarPtr* instance.

2. Clients should not be allowed to duplicate (copy) *CarPtr* instances, or to switch the car instance associated with a *CarPtr* instance.

3. The API of the *CarPtr* class should include both public member functions *operator*()* and *operator->()*.

4. You must store the five *CarPtr* instances in an array of length 5.

5. You must code classes *String* and *Date* yourself without relying on the Standard Template Library (STL) or other libraries.

6. Each car instance will contain references or pointers to multiple *String* instances, for instance, for the *Make* and *Model* of each car. However, the *Date* instance holding a car's *Date* field will be embedded directly in the car instance.

You must work alone on this project. Save all your code in a collection of header and code files and submit a zip archive with a (short) readme file containing instructions on how to use your VCS. Your code should compile under the GNU C++ compiler using the C++11 standard. No late submissions will be accepted.