CS 474: Object Oriented Programming Languages and Environments
Spring 2018

First C++ project

Due time: **9:00 pm on Monday 4/23/2018**

You are to implement an e-commerce site for selling household appliances, such as dishwashers, laundry machines, and clothes dryers. Your client is Ms. Rosey Sowpe, who owns an appliance shop. Your task for this project is to implement a proof-of-concept prototype in C++11. Your prototype will manage the appliances for sale in Ms. Sowpe's site. Each appliance contains the following fields:

1. *ID*—This is a unique (integer) identifier for each appliance.
2. *Type*—The type of an appliance is an enumeration with possible values of LAUNDRY_MACHINE, DISH_WASHER, and DRYER.
3. *Manufacturer*—This is a string denoting the company making the appliance (e.g., "Whirpool").
4. *Price*—This is the price of the product in US dollars and cents.
5. *Pictures*—An array of file paths containing a list of pictures for each appliance in an appropriate format (e.g., PNG, JPEG, etc.)

The last item above is an array of strings denoting file paths. Each file path identifies a file containing a byte sequence meant to display a picture of an appliance on the user's display. For the time being, you do not have to worry about creating, storing or displaying the images; 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 appliance instance expensive to store in memory.

To avoid cluttering the primary memory of your computer with images, you must store your appliance instances in files most of the time. Each such file will contain information a single appliance instance including its ID, Type, Manufacturer, etc. Also, you should implement a smart pointer class called *AppliancePtr* that dynamically loads instances of class *Appliance* when they are needed and stores them back on file when done with an appliance.

You are to code a test version of the program that contains just 10 appliances denoted by unique IDs 0 through 9. Each of the ten instances is stored in a separate file. At any point in time, only three of the appliance instances can be stored in memory; the remaining instances exist only in the ten files associated with your project.

When the program is started, the program creates a linked list of 10 appliance pointers; however, none of the appliance instances are loaded in memory. Next, appliances are loaded from memory and stored back to files (possibly after being modified) as needed while executing the commands described next. However, you should never hold more than 3 appliance instances in memory at all times.

Your project should support the command line interface below; no GUI is needed for this project. 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. l—List all appliances. Each appliance is loaded from disk and its information (ID, Type, etc.) is displayed in the user's screen, including the list of file paths, but with no images of the appliance. Appliance instances are stored back on disk and deleted as needed to keep a maximum of three total appliances in memory at all times.

2. `0 ... 9`—Edit a appliance. This command sets the numbered appliance to be the current appliance. If the appliance is not already in memory, it is loaded from disk and assigned to the corresponding appliance pointer. In this case, an appliance instance currently in memory may have to be stored back into its corresponding file and deleted from memory not to exceed the limit of 3 appliances.

3. `c`—Create an appliance. The user is prompted for the various fields in the appliance instance. The appliance is then stored to a file and associated smart pointer instance. The appliance's ID must be between 0 and 9. If an appliance with the same ID existed before this appliance is created, the older appliance is deleted from both memory and the associated file.

4. `p`—This command changes the price of the current appliance. The user is prompted for a new number, which becomes the new price of the appliance. The current appliance is not saved to file as part of this command. The new content of current appliance is displayed in the user's screen. The new price will be saved when the appliance instance is stored to its file.

5. `s`—Save all appliances. This command save the appliance(s) currently in memory to the corresponding file(s). First, the files are deleted (e.g., using the *remove()* function). Next, an *ofstream* instance is opened on the named file, and the appliance's information is saved to the new file. Finally, the stream is closed.

6. `q`—Quits the appliance manager.

Implementation notes. Your appliance manager must meet the following requirements. First, each appliance pointer should handle exactly one appliance. Second, clients (i.e., your project code) should not be allowed to create or delete appliance instances directly; these functions are handled by the smart pointer. Third, clients should not be allowed to duplicate (copy) smart pointers, or to switch the appliance instance associated with a a given smart pointer instance. Fourth, the API of the *AppliancePtr* class should include both *operator*()* and *operator->()*. Finally, you must code the linked list yourself, without relying on classes defined, e.g., in the Standard Template Library (STL) or other libraries.

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 Appliance Manager. The archive should also contain sample text files describing your initial appliances. Submit the archive by clicking on the link provided with this assignment. Your code should compile under the GNU C++11 compiler. No late submissions will be accepted.